MANUALE OPERATIVO QBASIC

ISTITUTO DI INFORMATICA
SAN PAOLO DI TORINO

MANUALE QBASIC

a cura di:

Istituto d'Informatica San Paolo di Torino s.r.l.

Via A.Boito 22 - 10154 Torino.

Tel. 011 / 247.32.00 (4 linee r.a.)

INDICE

INTRODUZIONE

Il Basic e' il linguaggio di programmazione piu' usato sui personal computer, grazie alla sua semplicita' di apprendimento e d'uso.

BASIC e' l'acronimo di:

Beginner's All-pourpose Symbolic Instruction Code.

Nel corso degli anni il Basic, da semplice linguaggio ad uso didattico (vedi CENNI STORICI), si e' evoluto fino all'attuale Quick Basic, arricchendosi di istruzioni e strutture logiche che lo rendono un linguaggio potente e professionale, pur mantenendo le caratteristiche iniziali di semplicita' e immediatezza di impiego.

Queste sue particolarita' lo rendono adatto sia ai professionisti che a quella vasta utenza di personal computer che non fa della programmazione una professione.

CENNI STORICI

La prima versione di Basic e' stata messa a punto da J. G. Kemeny e T. E. Kurtz, professori dell'universita' di Dartmouth, con l'aiuto di un gruppo di studenti, nel 1964.

Il nuovo linguaggio doveva servire all'alfabetizzazione informatica di tutti gli studenti, e i linguaggi in uso fino a quel momento erano troppo tecnici (Assembler) o troppo specialistici (Fortran, Algol) per poter essere utilizzati, a questo scopo, in breve tempo, da principianti.

Inoltre, invece di compilatori viene utilizzato, con il nuovo linguaggio, un interprete, che permette l'interattivita' utente-elaboratore, agevolando cosi' la stesura e la messa a punto dei programmi.

Il Basic si arricchisce successivamente di istruzioni che permettono una piu' ampia gestione dei file, il miglioramento dell'interattivita' e la possibilita' di gestire *subroutine* e *moduli.*

Con l'avvento dei personal computer il Basic esce dal ristretto mondo della didattica per diventare il linguaggio ufficiale di questa nuova realta' che ha portato l'informatica al livello di diffusione che tutti conosciamo.

Con il Quick Basic si e' ulteriormente semplificato

l'uso del linguaggio grazie ad un EDIT particolarmente potente e versatile, all'introduzione di nuove istruzioni che facilitano la strutturazione dei programmi, all'utilizzo di etichette (*label*), invece di numeri di riga, che migliorano la leggibilita' dei programmi, ad una Guida in linea particolarmente efficace e a molte altre funzionalita' che impareremo presto a conoscere.

E' importante sottolineare la completa compatibilita' di Quick Basic con le precedenti versioni Microsoft GW-Basic e IBM Basica, che permette la gestibilita' nel nuovo ambiente dei vecchi programmi.

I LINGUAGGI DI PROGRAMMAZIONE

Fino a circa venti anni fa, quando computer molto meno potenti dei moderni PC occupavano intere stanze, il Software veniva sviluppato da un numero molto ristretto di specialisti, che, dopo studi lunghi e difficili, riuscivano a trasmettere le informazioni nella memoria di questi grandi calcolatori, grazie all' uso di schede perforate e relativi dispositivi di lettura e scrittura, capaci di trasformare le perforazioni in informazioni interpretabili dal computer.

L' unico LINGUAGGIO, se cosi' possiamo definirlo, era quello del microprocessore e si snodava fra alcune centinaia di *chiamate* alle funzioni elementari della CPU, espresse in codice ESADECIMALE, caratteristico dell' Assembler (detto anche Linguaggio Macchina).

Questa codifica era formata non da parole. ma da numeri in base 16.

Possiamo quindi immaginare la difficolta' del suo utilizzo.

Con queste premesse, diventa facile dedurre che fare il programmatore in quell'epoca non era una impresa semplice.

Con lo sviluppo della miniaturizzazione dei materiali, anche i computer diventano piu' piccoli e sempre piu'

potenti.

Il loro utilizzo, inoltre, diventa ben presto un'esigenza irrinunciabile, a tal punto da spingere le societa' che si occupano di software a cercare soluzioni che permettano anche ai meno esperti un facile approccio alle problematiche della produzione del *software applicativo.*

Nasce cosi' il concetto di LINGUAGGIO di ALTO LIVELLO, orientato piu' al problema da risolvere che alla macchina; le istruzioni sono composte da parole anziche' da simboli matematici.

Il capostipite di questa nuova generazione di linguaggi e' senz'altro il COBOL, la cui formulazione e' di tipo discorsivo: impiega cioe' termini e punteggiatura nel rispetto delle regole sintattiche, proprio come un linguaggio parlato.

In seguito si svilupparono numerosi altri linguaggi, orientati e specifici di certi settori specialistici del mondo lavorativo.

Fra questi ultimi possiamo annoverare il BASIC e il PASCAL, universalmente riconosciuti come linguaggi completi (utilizzati per risolvere un gran numero di problematiche sia di tipo scientifico che gestionale) e al tempo stesso abbastanza accessibili, tanto da rendere agevole anche ai neofiti l'approccio al mondo della programmazione informatica.

Il PASCAL e' stato col tempo completato e migliorato, tanto da rappresentare oggi la testa di serie della programmazione strutturata: esso e' diventato un linguaggio molto articolato, ma purtroppo sempre piu' difficile da impiegare, anche per i piu' esperti (come accade spesso in informatica quando si aggiungono troppe funzionalita'...).

Il BASIC e' invece il linguaggio che, pur rispettando le caratteristiche di completezza e versatilita', ha mantenuto e migliorato nel tempo l'originale facilita' di impiego, per cui e' diventato il linguaggio piu' diffuso, in particolare sui personal computer.

La societa' che ha piu' di ogni altra migliorato e completato il BASIC e' senz'altro la Microsoft, con edizioni sempre piu' potenti del Quick Basic, che viene fornito in una versione semplificata ma non per questo meno potente, insieme al sistema operativo MS-DOS, come file eseguibile chiamato QBASIC.EXE.

La versione a cui QBasic fa riferimento e' la 4.5, oggi diventata lo standard mondiale di programmazione ed impegata anche sotto Windows, dove, arricchita con funzioni grafiche particolarmente efficaci, e' diventata VISUAL BASIC.

Qesto manuale descrive l' impiego del QBasic, senza trascurare gli eventuali richiami alla versione completa del pacchetto (il QuickBasic 4.5), che permette, in piu' del QBasic, la ***compilazione*** dei programmi sorgente, come vedremo di seguito; inoltre ha qualche opzione

di menu' in piu'.

LA COMPILAZIONE

Un programma e' formato da una sequenza di istruzioni composte da parole, simboli di punteggiatura, operatori aritmetici, ecc., uniti fra loro secondo certe regole formali.

Queste istruzioni, quando si esegue il programma (RUN) , sono ad una ad una tradotte da un *interprete* in linguaggio di macchina ed eseguite.

Il QBasic, di cui l'*interprete* e' una delle funzionalita', si presenta sullo schermo con una barra di menu', simile a quella delle applicazioni Windows, da cui e' possibile scegliere una serie di opzioni che permettono di attivare le relative funzioni, come vedremo in dettaglio di seguito.

Un programma scritto in Basic (*sorgente*) puo' essere eseguito soltanto in presenza del QBasic, per mezzo dell'*interprete.*

Per trasformare un programma *sorgente* in uno *eseguibile* (attivabile direttamente dal *prompt* del DOS), occorre sottoporlo al COMPILATORE.

Si tratta di un programma traduttore, che non esegue le istruzioni ma le traduce tutte in linguaggio di macchina (codifica esadecimale), controllando gli eventuali errori di sintassi e producendo, se tutto e' ok, un file **.EXE** , che viene salvato su disco.

Come gia' detto, il *compilatore* e' l'unica parte mancante nel modulo QBasic fornito con il sistema operativo MS-DOS.

L' AMBIENTE DI PROGRAMMAZIONE

Per attivare il QBasic, basta scriverne il nome dal prompt del DOS:

C:\> QBASIC.

E' opportuno, specialmente all'inizio, attivare QBasic da una subdirectory, ad esempio da **C:\LAVORI>**, che raccogliera' anche tutti i nostri programmi.

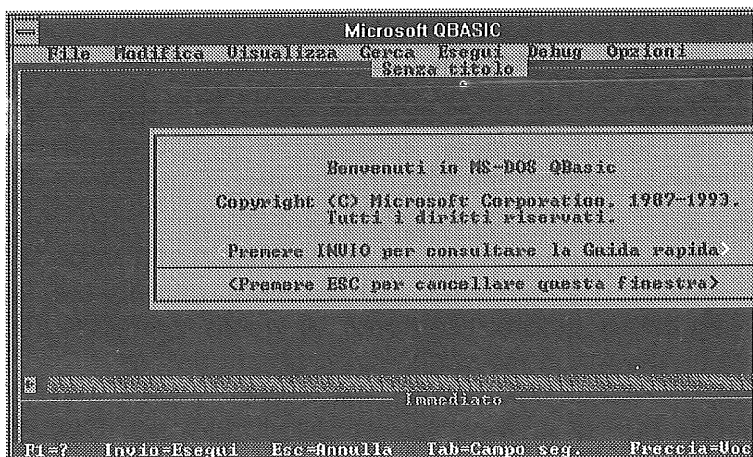
Ogni programma Basic, salvato su disco dopo averlo digitato, e' un **file ASCII**, con estensione **.BAS** aggiunta automaticamente al nome primario del file che invece viene scelto dal programmatore.

Diversa e' invece la modalita' con cui GW-Basic, predecessore del QBasic, salva i file sorgenti: essi vengono compressi durante la registrazione su disco (allo scopo di risparmiare spazio), per essere decompressi quando vengono richiamati in memoria.

Questo spiega perche', a volte, non e' possibile richiamare da QBasic alcuni file .BAS presenti su disco; si tratta, infatti, di sorgenti creati con GW-Basic.

Per poter gestire questi programmi nel nuovo ambiente QBasic, occorre salvarli in GW-Basic con l'opzione ***A*** (ASCII) prima di importarli.

Il nuovo ambiente si presenta sullo schermo con la

finestra riprodotta sotto, recante al centro una **finestra di dialogo** che invita a premere **INVIO**, per accedere alla **Guida in Linea**, oppure **ESC** per iniziare il lavoro.

LA FINESTRA QBASIC

Vediamo in dettaglio, a partire dall'alto, le parti costitutive della finestra QBasic, che esamineremo di seguito voce per voce:

- La Barra del titolo (**Microsoft QBASIC**), con a sinistra la **casella del menu' di controllo**;

- La Barra dei menu', che riporta una serie di voci (**File,Modifica**, ecc.);

- L' Area di lavoro, che comprende la maggior parte dello schermo e serve per inserire le righe di programma;

- Una seconda Area di inserimento (**Immediato**), dove e' possibile inserire gruppi limitati di comandi;

- Le Barre di scorrimento orizzontali e verticali, per accedere a parti del documento non visibili sullo schermo;

- La Barra delle chiavi che riporta alcuni abbinamenti tra tasti funzione e comandi.
  Sulla destra viene riportata la posizione corrente del cursore in numero di riga e colonna.

Si puo' accedere in qualunque punto della finestra, attivando funzioni con il **Mouse** oppure utilizzando la tastiera.

I TASTI DI EDIT

Sono cosi' definiti alcuni tasti che attivano operazioni che permettono di spostarsi velocemente all'interno di un listato e altri, detti *tasti-scorciatoia*, che consentono interventi rapidi di modifica del testo

Esaminiamo di seguito i tasti e le combinazioni di tasti utilizzati a questo scopo:

INS Commuta tra le due modalita' di scrittura *Inserimento e Sovrapposizione;*

HOME E' anche indicato con una freccia obliqua e serve per spostarsi a *Inizio riga;*

END E' anche indicato con FINE e va a *Fine riga;*
una riga puo' essere piu' lunga della parte visibile dello schermo;

PG UP
PG DN Sono anche indicati con PAG SU' e PAG GIU' e servono a scorrere il testo del programma di una pagina video alla volta;

CTRL+Frecce orizzontali Permette di saltare da una parola ad un'altra (da spazio a spazio);

CTRL+HOME Permette di spostarsi all' inizio del testo;

CTRL+FINE Permette di spostarsi alla fine del testo.

L' uso accorto ed esperto di queste scorciatoie rende agevole la messa a punto e velocizza gli interventi di correzione.

I TASTI DI SELEZIONE

In QBasic si puo' utilizzare l'Area di memoria **Appunti**, come in ambiente Windows, per **SPOSTARE, COPIARE e INCOLLARE** parti di programma, ma prima occorre indicare su quale parte di documento agire.

Occorre cioe' **SELEZIONARE** una parola, una riga o molte righe, poiche' solo dopo aver effettuato una selezione possiamo attivare gli Appunti (presenti nel Menu' **Modifica**, come in Windows).

La SELEZIONE deve sempre essere continua; non si possono selezionare, ad esempio, le prime tre righe, saltare la quarta e quindi riprendere la selezione dalla quinta alla decima riga.

Per effettuare una selezione occorre *puntare* con il **Mouse** all' inizio della selezione, premere e tenendo premuto il pulsante sinistro, *trascinare* il mouse lungo la selezione da evidenziare, che apparira' in *reverse* rispetto al testo non selezionato.

Da tastiera, il tasto che attiva la selezione e' **SHIFT** (o **MAIUSC**), che usato insieme ai cursori (i **tasti freccia+**) permette di evidenziare una parola, una riga o piu' righe.

Prima di premere SHIFT, occorre naturalmente

spostare il cursore all' inizio della zona da selezionare, affinche' QBasic sappia da dove iniziare.

Per **DESELEZIONARE** si preme un tasto qualunque, oppure si fa **CLICK** con il Mouse in una zona fuori selezione.

Se non si effettua l'operazione di deselezione, quando si riprende a digitare, tutto il testo selezionato verra' cancellato.

La figura che segue mostra l'aspetto di una selezione:

La selezione rende attive le opzioni del menu' ***Modifica***, che permettono di spostare, copiare e incollare testo (**Taglia, Copia, Incolla**).

LE FINESTRE DI DIALOGO

Durante le operazioni di *Edit* il QBasic dialoga con l'operatore inviando **messaggi**, che possono essere di *errore* o di *segnalazione* (ad esempio: "IL FILE NON E' STATO SALVATO") o di altro tipo.

Vengono inviati messaggi anche per richiedere ulteriori informazioni, senza le quali QBasic non puo' proseguire (ad esempio quando si salva per la prima volta un programma viene richiesto il nome).

In tutti questi casi viene visualizzata una *FINESTRA DI DIALOGO*, i cui contenuti sono diversi a seconda del tipo di segnalazione o richiesta:

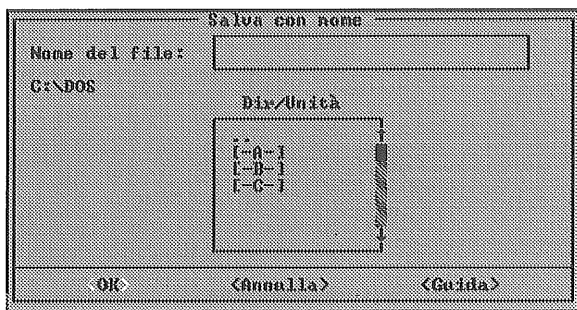
- Se la finestra e' di segnalazione, conterra' una frase esplicativa e dei pulsanti (del tipo OK, ANNULLA, SI, NO, ecc.), che possono essere attivati sia da tastiera, usando TAB per spostarsi tra i vari pulsanti e confermando quello prescelto con INVIO, sia con il mouse facendo CLICK sul pulsante desiderato.

- Se la finestra di dialogo sollecita *informazioni aggiuntive*, occorre scrivere le informazioni richieste nelle apposite caselle, quindi attivare il tasto OK per confermarle o il tasto ANNULLA per annullarle.

NOTA

Con il tasto funzionale ESC e' possibile chiudere qualunque finestra di dialogo, con un'operazione simile ad ANNULLA, anche quando non e' presente questo pulsante.

Anche tutte le voci di menu' che terminano con i puntini di sospensione aprono una finestra di dialogo che richiede informazioni aggiuntive.

Nella figura successiva vi e' un esempio di finestra di dialogo: viene visualizzata quando si seleziona la voce *Salva con nome* del menu' *File*:

Nel prossimo capitolo sono descritti i *menu' a tendina*, utile complemento alla stesura e conservazione dei nostri lavori, che sono attivati dalle voci della Barra dei menu'.

LA BARRA DEI MENU'

Le voci che compongono la Barra dei Menu' sono anche chiamate *Menu' a tendina*, poiche' attivandole con il *click* del mouse viene visualizzata una tendina, contenente varie opzioni selezionabili anch'esse con un *click*.

Le voci di menu' si possono attivare anche con la tastiera (ALT + *Lettera Evidenziata)*, ma e' senz'altro piu' comodo l'utilizzo del mouse.

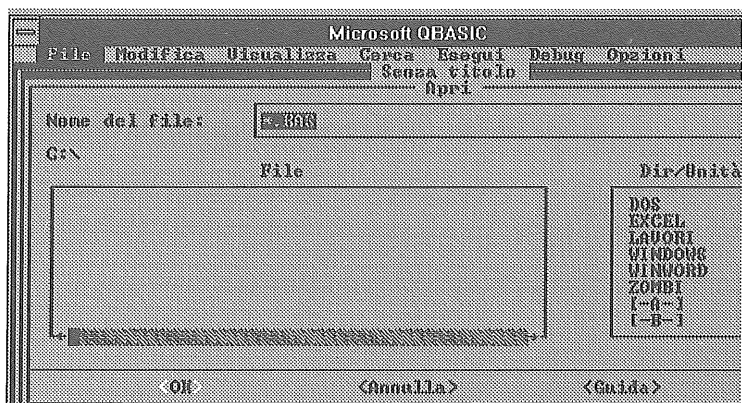
Le opzioni che sono seguite dai punti di sospensione aprono una finestra di dialogo, che consente di definire ulteriori parametri.

Vediamo di seguito le varie voci riportate sulla Barra dei menu'.

- **MENU' FILE**

  - **NUOVO** Azzera il foglio di programmazione.

Se il listato precedente non e' stato salvato, il sistema sollecita questa operazione prima di azzerare.
Rispondere OK per salvare, NO per non salvare, ANNULLA per annullare le scelte precedenti

- **APRI** Carica un file .BAS, selezionato nella finestra di dialogo dedicata alla gestione dei file, che viene visualizzata.

Si puo' fare *click* sul nome prescelto oppure scriverlo direttamente nella casella di testo *Nome del file.*
Di fianco all' elenco dei file .BAS troviamo il box di elenco delle Directory e dei Disk driver disponibili, dove e' possibile scegliere l'unita' e la subdirectory desiderata.

- **SALVA** Aggiorna un file che gia' aveva un nome (revisioni di un listato successive alla prima stesura...).
Se il file non ha ancora un nome, questa scelta si comporta come la successiva.

- **SALVA CON NOME** Apre una finestra di dialogo identica a quella relativa alla voce APRI, ma in questo caso e' necessario indicare il nome del file nella apposita casella.
L'estensione .BAS e' aggiunta automaticamente dal sistema.

- **STAMPA** Invia su stampante il listato completo, oppure la sola parte selezionata precedentemente.

- **ESCI** Disattiva il QBasic.
Questa opzione attiva una finestra di dialogo in cui si chiede, se il file e' cambiato dall' ultimo salvataggio, se si vuole salvarlo prima di uscire.
Anche in questo caso possiamo scegliere SI, NO oppure ANNULLA il comando.

• MENU' MODIFICA

- **TAGLIA** Trasferisce nell'area di memoria *Appunti* il testo selezionato, eliminandolo dal listato, come se lo avessimo ritagliato con le forbici.
La parte di testo cosi' tagliata potra' essere in seguito *incollata* a partire dalla posizione del cursore, con l'opzione INCOLLA.

- **COPIA** Funziona come TAGLIA, ma non elimina dal listato il testo selezionato.

- **INCOLLA** Porta sul listato (*Incolla*) il contenuto dell'area Appunti, a partire dalla posizione del cursore.

- **NUOVA SUB** Questa scelta, insieme a NUOVA FUNCTION, fa parte delle scelte avanzate.
Apre una finestra di dialogo, dove si

scrive il nome della SUB che si intende creare, che diventera' un file separato, legato al modulo principale a livello logico ma non a livello fisico.

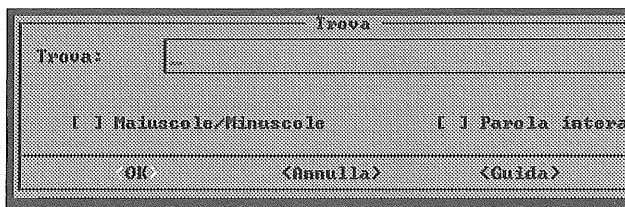
- **NUOVA FUNCTION** Crea una funzione. Si opera come per NUOVA SUB. Una funzione utilizza variabili che vengono passate dal modulo principale, restituendole al ritorno con le modifiche apportate.

- **MENU VISUALIZZA**

- **SUBS** Permette di spostare la visualizzazione (e l'attivita') sulla *Sub* indicata, e di tornare al modulo principale. La finestra di dialogo che viene visualizzata permette inoltre di eliminare una Sub, semplicemente selezionandola e quindi attivando il pulsante CANCELLA.

- **DIVIDI** Permette di visualizzare sul video due parti di listato.

- **SCHERMO OUTPUT** Permette la visualizzazione della schermata lasciata dal programma eseguito. E' molto utile per verificare la posizione dei dati a per la sistemazione delle

mappe.
Questa opzione si attiva facilmente anche da tastiera con il tasto funzionale *F4*.

- **MENU' CERCA**

- **TROVA** Effettua la ricerca di una parola (variabile o etichetta) all' interno di un listato o di una Sub.
Con l'opzione *Parola intera* si richiede di ricercare tutte le ricorrenze della parola indicata.
Senza questa opzione saranno evidenziate tutte le sequenze di lettere uguali a quella indicata, escludendo gli spazi (ad esempio:
trova PENA trovato **OPEN A$**).

- **RIPETI TROVA** Riprende la ricerca sulla parola indicata in TROVA, fino alla prossima ricorrenza.

- **CAMBIA** Sostituisce la ricorrenza indicata nella casella *Trova* con quella indicata nella casella *Cambia in.*
Si puo' inoltre indicare se operare in automatico (*Cambia tutto*) oppure se si vuole confermare la sostituzione ad ogni ricorrenza trovata (*Trova e verifica*)

Cambia

Trova:

Cambia in:

[ ] Maiuscole/Minuscole [ ] Parola intera

< Trova e verifica > <Cambia tutto> <Annulla> <Guida>

- **MENU' ESEGUI**

- **AVVIA** Esegue il programma editato iniziando dalla prima istruzione eseguibile.

- **RIAVVIA** Cancella qualsiasi dato inserito durante una precedente esecuzione ed evidenzia la prima istruzione eseguibile.

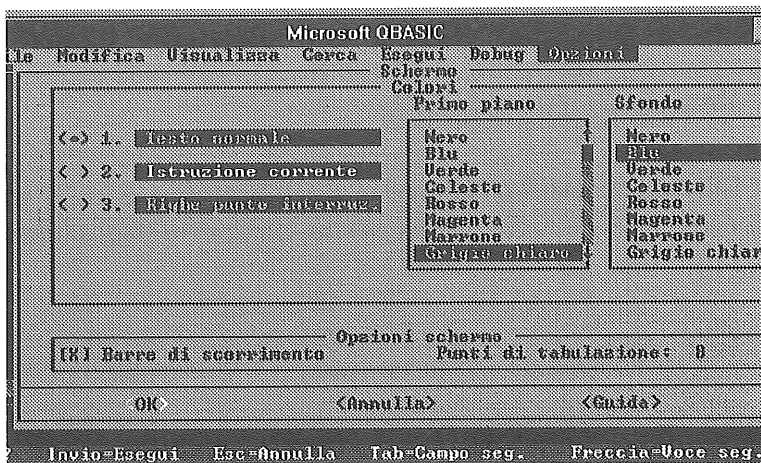
- **CONTINUA** Riavvia un programma quando e' terminato senza azzerare i campi o

continua l'esecuzione di un programma interrotto (ad esempio con CTRL+BREAK).

- ## MENU' DEBUG

Le voci di questo menu' permettono, entro certi limiti, di verificare i punti esecutivi del listato, durante il test del programma: si tratta di opzioni avanzate, che presuppongono ottime conoscenze sulla struttura del linguaggio.

- ## MENU OPZIONI

- **SCHERMO** Apre una grande finestra di dialogo

con la quale si possono modificare i colori dell'ambiente e la visualizzazione delle barre di scorrimento, nonche' reimpostare l'ampiezza di tabulazione.

- **PERCORSO DI GUIDA** Consente di cambiare il percorso DOS su cui rintracciare i file di aiuto (*.HLP*), normalmente posizionati nella stessa directory da cui si attiva QBasic.

- **VERIFICA SINTASSI** Con questa opzione si puo' attivare o disattivare la verifica sintattica.
  Questa opzione, quando e' attiva (evidenziata da un pallino di spunta davanti alla voce), controlla che la sintassi dell'istruzione appena scritta sia corretta, segnalando l'errore, quando rilevato, e indicando con il cursore, per quanto possibile, il punto in cui si e' verificato.

ELEMENTI DEL LINGUAGGIO

Le istruzioni Basic sono formate da un insieme di parole e simboli che vengono definiti ELEMENTI DEL LINGUAGGIO, che possiamo classificare come indicato di seguito.

- **INSIEME DEI CARATTERI**

  L'insieme dei caratteri utilizzati dal Basic comprende i caratteri alfabetici maiuscoli e minuscoli, i numeri da 0 a 9, e i seguenti caratteri speciali, di cui indichiamo l'uso:

  - **SUFFISSI PER TIPO DI DATI**

    ! Punto esclamativo: valore numerico a semplice precisione;

    # Cancellino: valore numerico a doppia precisione;

    $ Dollaro: dato a stringa;

    % Percento: valore numerico intero;

    & E commerciale: valore numerico intero LONG.

- OPERATORI MATEMATICI

- `*` Asterisco: simbolo di moltiplicazione;
- `-` Segno meno: simbolo di sottrazione o segno algebrico;
- `/` Barra: segno di divisione;
- `=` Uguale: operatore relazionale o simbolo di assegnazione;
- `>` Segno di maggiore: operatore relazionale;
- `+` Segno piu': simbolo di somma;
- `.` Punto: simbolo di punto decimale;
- `<` Segno di minore: operatore relazionale;
- `\` Barra rovesciata: simbolo di divisione intera;
- `^` Accento circonflesso: simbolo di elevazione a potenza;
- `( )` Parentesi: racchiudono parti di una espressione matematica e operatori di funzione.

- **CARATTERI SPECIALI**

' Virgolette semplici: indicano una frase di commento;

; Punto e virgola: controlla l'output delle istruzioni PRINT e INPUT;

, Virgola: controlla l'output delle istruzioni PRINT e INPUT;

: Due punti: separa piu' istruzioni all'interno di una riga;

? Punto interrogativo: prompt dell'istruzione INPUT o abbreviazione di PRINT;

" Virgolette doppie: delimitatore di stringa.

- **PAROLE RISERVATE**

Sono le parole del linguaggio Basic che esprimono istruzioni e comandi (PRINT, GOSUB, KILL), funzioni (COS,LEN,SQR), parametri e specifiche (SHARED, USING, STEP), operatori (AND, OR, AS), dichiarazioni di dati (DIM, DATA).

Le parole riservate non possono essere usate come nomi di variabili; debbono essere separate

dagli altri elementi del linguaggio da uno spazio o da un carattere speciale consentito dalla sintassi.

Nel *Sommario* vengono riportate in maiuscolo.

- **COSTANTI**

Sono grandezze fornite direttamente nelle istruzioni Basic, che non cambiano valore durante l'esecuzione del programma.

Vi sono due tipi di costanti: *numeriche* e *a stringa*.

Le costanti numeriche possono contenere:

- le cifre da 0 a 9;
- gli operatori algebrici - e + (facoltativo);
- il punto decimale (es. 0.524, -8432.12).

Le costanti a stringa possono contenere qualunque carattere e sono delimitate da virgolette doppie; possono contenere fino a 32.767 caratteri e anche se sono composte da sole cifre non possono essere utilizzate per operazioni matematiche.

- **VARIABILI**

Indicano i campi di memoria che vengono utilizzati

per contenere i dati durante l'esecuzione del programma.

Il nome di una variabile puo' essere lungo al massimo 40 caratteri, puo' contenere lettere e numeri, ma deve iniziare con una lettera; l'ultimo carattere e' in genere il simbolo che ne determina il tipo.

Non e' consentito utilizzare parole riservate come nomi di variabili.

Il carattere utilizzato come suffisso classifica la variabile in uno dei seguenti tipi:

- **STRINGA** (suffisso:$)

  Contiene caratteri di qualunque tipo da un minimo di 0 a un massimo di 32.767.

  Esempio: Nome$, A10$

- **NUMERICO INTERO** (suffisso:%)

  Occupa 2 byte di memoria; puo' contenere valori interi nei limiti -32.768, +32.767.

  Esempio: I%, contatore%.

- **NUMERICO INTERO LONG** (suffisso:&)

Occupa 4 byte di memoria; puo' contenere valori interi nei limiti:
-2.147.483.648, 2.147.483.647.

Esempio: Intero&.

- **NUMERICO IN SINGOLA PRECISIONE** (suffisso:!)

  Occupa 4 byte di memoria; puo' contenere valori interi o decimali fino a 7 cifre.

  Esempio: Quantita!, ABC.

- **NUMERICO IN DOPPIA PRECISIONE** (suffisso:#)

  Occupa 8 byte di memoria; puo' contenere valori interi o decimali fino a 15 cifre.

  Esempio: Importo#.

Se viene omesso il suffisso la variabile viene assunta in semplice precisione.

E' possibile evitare l'uso del suffisso per identificare il tipo di variabile, utilizzando l'istruzione DEF *tipo* che permette di definire le variabili in base alla lettera iniziale del nome (vedi

Sommario).

Vediamo, ad esempio, come si puo' codificare la struttura di un record relativo ad un archivio articoli:

```
TYPE Articolo
     Codice AS INTEGER
     Descrizione AS STRING * 30
     Giacenza AS SINGLE
     Prezzo AS DOUBLE
```

- **VETTORI E MATRICI**

I vettori e le matrici (tabelle a una o piu' dimensioni)) sono delle particolari strutture di dati in cui le variabili elementari che li compongono sono tutte dello stesso tipo e nome e vengono chiamate *elementi*.

Ogni elemento e' individuato dal *nome* seguito dal *subscritto*.

Il subscritto e' composto da uno o piu' valori numerici interi (*indici*), racchiusi fra parentesi, che indicano la posizione dell'elemento nella tabella: *zero* rappresenta la prima posizione e cosi' via.

Il numero di indici utilizzati nel subscritto dipende dalle dimensioni della tabella.

Le tabelle ad una dimensione vengono dette *vettori*, quelle a piu' dimensioni *matrici*.

Il numero massimo di dimensioni consentito e' 60.

Vettori e matrici vengono definiti con l'istruzione DIM (vedi *Sommario*).

Se una tabella non viene dimensionata assume una struttura di 11 elementi per ogni dimensione (indice massimo di subscritto uguale a 10).

Vediamo alcuni esempi:

```
DIM N$(50)          ' definisce un vettore con 51
                    ' elementi di tipo stringa

N$(5) = "Rossi"     ' assegna al terzo elemento
                    ' la costante "Rossi"

DIM A%(20, 20)      ' definisce una matrice a 2
                    ' dimensioni con elementi
                    ' di tipo numerico intero

PRINT A%(10,10)     ' scrive l'elemento con
                    ' coordinate 10, 10
```

- **OPERATORI DI RELAZIONE**

Servono a confrontare tra loro due valori espressi

da costanti, variabili o espressioni a formare una *condizione*, utilizzata normalmente nelle istruzioni condizionali.

Vediamo di seguito i simboli utilizzati e il loro significato:

=	uguale a;
<> o ><	diverso da;
<	minore di;
>	maggiore di;
=< o <=	uguale o minore di;
=> o >=	uguale o maggiore di.

Il risultato di un confronto viene normalmente utilizzato per prendere una decisione riguardo al flusso di esecuzione di un programma e puo' essere *vero (1)* o *falso (0)*.

Il confronto puo' essere fatto tra dati omogenei (numerici di qualunque tipo o a stringa).

Il confronto tra dati a stringa avviene da sinistra verso destra e carattere per carattere.

Il valore di ogni carattere e' determinato dalla

tabella dei codici ASCII, in base alla quale la lettera *A* e' minore della lettera *B* e cosi' via, le lettere minuscole sono maggiori delle maiuscole, i numeri e i caratteri speciali sono minori delle lettere, lo spazio (*blank*) e' il valore piu' basso.

- **LINEE DI PROGRAMMA E ETICHETTE**

Una linea di programma puo' contenere fino a 255 caratteri, inclusi gli spazi.

Si possono scrivere piu' istruzioni sulla stessa riga separandole con il carattere *due punti* (:).

Poiche' l'Edit del Basic visualizza sullo schermo 78 caratteri per riga, occorre utilizzare lo scorrimento orizzontale (freccia destra e sinistra) per visualizzare righe piu' lunghe.

Per la migliore leggibilita' del programma si consiglia di scrivere, per quanto possibile, righe che non superino le dimensioni dello schermo e delle normali stampanti a 80 colonne.

A volte e' necessario identificare un punto del programma per effettuare, ad esempio, un passaggio di controllo tramite le istruzioni GOTO e GOSUB.

Nelle precedenti versioni del Basic (GW-Basic, Basica) ogni linea di istruzione era preceduta da

un numero di riga che veniva utilizzato come identificatore (esempio: GOTO 120).

Poiche' in QBasic non e' necessaria la numerazione di riga, viene utilizzato, ove occorre, un identificatore alfanumerico detto <u>*etichetta*</u>: e' un nome del programmatore che, come i nomi delle variabili, puo' essere formato da un numero di caratteri alfanumerici compreso tra 1 e 40, deve iniziare con una lettera e terminare con il carattere *due punti* (:); non si possono utilizzare come etichette parole riservate ed e' indifferente l'uso di lettere maiuscole e minuscole.

Quando in una istruzione viene utilizzato il nome di una etichetta si debbono omettere i due punti di chiusura.

Vediamo un esempio:

```
Inizio:
  .
  .
  .
GOTO Inizio
```

STRUTTURE DI CONTROLLO

In questo capitolo verranno trattati quei gruppi di istruzioni che permettono di modificare il normale ordine sequenziale di esecuzione delle istruzioni di un programma.

Permettono di eseguire operazioni ripetitive per un certo numero di volte o fino al verificarsi di un certo evento, di creare diramazioni nel flusso esecutivo di un programma in seguito al verificarsi di una certa condizione, di passare il controllo ad altro punto del programma.

• GOTO

Esegue un salto incondizionato alla riga specificata in argomento da una etichetta o da un numero di riga (per precedenti versioni di Basic).

La riga a cui viene passato il controllo deve trovarsi nella stessa procedura o subroutine della istruzione GOTO.

Sebbene sia una istruzione molto semplice, si consiglia di utilizzarla il meno possibile, preferendole istruzioni di controllo strutturate, che trattiamo di seguito, poiche' l'utilizzo di molte istruzioni GOTO in un programma ne rende

difficoltosa la lettura e la manutenzione.

Esempio:

```
CLS
 .
 .
 .
Ripeti:
 .
 .
 .
GOTO Ripeti
```

- **GOSUB...RETURN**

Passa il controllo alla subroutine specificata in argomento da una etichetta o da un numero di riga (per precedenti versioni di Basic).

Il ritorno dalla subroutine avviene tramite l'istruzione RETURN: se priva di argomento il controllo ritorna all'istruzione successiva a quella contenente GOSUB; se si vuole ritornare ad un punto diverso del programma occorre specificarne in argomento l'etichetta o il numero di riga.

Una subroutine puo' essere eseguita quante volte si desidera in un programma e puo' contenere piu' istruzioni RETURN.

Le subroutine possono essere scritte in qualunque punto del programma, ma si consiglia, per una migliore leggibilita', di inserirle dopo il programma principale.

Una subroutine deve essere sempre preceduta da una istruzione che interrompe il normale flusso di esecuzione (GOTO, END, STOP), in modo da non eseguirla accidentalmente, ma soltanto con una chiamata GOSUB.

Esempio:

```
    .
    .
    .
GOSUB Calcola     'salta alla
    .            'subroutine Calcola
    .
    .
Calcola:
    .
    .
    .
RETURN     'ritorna all'istruzione
           'successiva al GOSUB
```

- **IF...THEN...ELSE**

Struttura di decisione che permette di eseguire in alternativa due flussi di esecuzione in funzione dell'analisi della condizione posta come argo-

mento della IF: se la condizione e' *vera* vengono eseguite le istruzioni che seguono la clausola THEN, se e' *falsa* quelle che seguono la clausola ELSE.

Se non viene specificato ELSE, per condizione falsa l'esecuzione prosegue dalla riga successiva alla istruzione IF...THEN.

Si puo' scegliere tra due forme di impostazione della IF:

- **MONORIGA** Tutte le istruzioni relative alle clausole THEN e ELSE devono essere scritte sulla stessa riga; va bene quando il numero di istruzioni e' ridotto, altrimenti si pregiudica la leggibilita' del programma; se poi le IF sono nidificate (IF dentro un'altra IF) l'operazione diventa estremamente complessa e la codifica e' pressoche' illeggibile.

- **STRUTTURA A BLOCCO** Permette di scrivere su piu' righe le istruzioni relative alle clausole THEN e ELSE, in modo da poter utilizzare in modo chiaro strutture complesse.

Esempi:

```
REM  formato monoriga
IF B <> 0 THEN X = A / B ELSE PRINT "errore"
```

```
REM  struttura a blocco
IF B <> 0 THEN
  X = A / B
ELSE
  PRINT "operazione impossibile"
ENDIF
```

- **SELECT CASE**

Struttura di decisione a scelte multiple.

L'argomento che segue SELECT CASE puo' essere una variabile o una espressione numerica o a stringa.

Gli argomenti delle clausole CASE (costanti, variabili o espressioni) debbono essere dello stesso tipo e possono indicare un singolo valore (CASE 1), un intervallo di valori (CASE 1 TO 5) o un elenco di valori (CASE 1,3,5,7,9).

Esempi:

```
SELECT CASE Scelta$
    CASE "A"
        CALL TipoA
    CASE "B"
        CALL TipoB
    CASE "C"
        CALL TipoC
```

```
    CASE ELSE
        PRINT "Scelta errata"
END SELECT


SELECT CASE n
    CASE 0,2,4,6,8
        PRINT "La cifra e' pari"
    CASE 1,3,5,7,9
        PRINT "La cifra e' dispari"
    CASE ELSE
        PRINT "Valore > 9 o negativo"
END SELECT
```

- **ON...GOSUB|GOTO**

  Istruzione di decisione, molto simile alla precedente, ma con qualche limitazione:

  - L'argomento che segue ON puo' essere soltanto di tipo numerico intero positivo (MAX 255) che indica quale GOSUB|GOTO attivare (valore posizionale);

  - permette soltanto di attivare istruzioni GOSUB o GOTO;

  - non gestisce intervalli di valori o valori multipli.

Esempio:

```
ON scelta% GOSUB Sub1,Sub2,Sub3,Sub4
 .
 .
 .
```

- **FOR...NEXT**

Esegue il blocco di istruzioni contenute tra FOR e NEXT un numero di volte determinato da un contatore che si incrementa o decrementa da un valore iniziale a un valore finale.

Con l'opzione STEP e' possibile definire l'incremento del contatore ad ogni ciclo; se il valore dell'argomento di STEP e' negativo il contatore si decrementa; se viene omesso STEP il contatore si incrementa di 1.

Esempio:

```
REM     Cornice di asterischi
FOR i% = 1 TO 80
LOCATE (1, i%): PRINT "*"
NEXT
FOR i% = 2 TO 23
LOCATE ( i%, 80): PRINT "*"
NEXT
FOR i% = 79 TO 1 STEP -1
LOCATE (23, i%): PRINT "*"
```

```
NEXT
FOR i% = 22 TO 2 STEP -1
LOCATE (i%, 1): PRINT "*"
NEXT
```

- **WHILE...WEND**

Questa struttura controlla l'esecuzione di un blocco di istruzioni ripetendola non un numero predefinito di volte come FOR...NEXT, ma fino a quando la condizione impostata come argomento resta vera.

Esempio:

```
WHILE r$ <> "S" AND r$ <> "N"
    PRINT "Rispondere S o N"
    INPUT r$
WEND
```

- **DO...LOOP**

Struttura di controllo molto simile alla precedente, ma piu' versatile.

Puo' infatti eseguire il test di condizione sia all'inizio che alla fine del ciclo, poiche' la condizione puo' essere inserita come argomento sia della proposizione DO che della LOOP.

Si puo' inoltre far terminare il ciclo sia per condizione *vera* (UNTIL) che per condizione *falsa* (WHILE).

Esempi:

```
REM  Esempio precedente
DO WHILE r$ <> "S" AND r$ <> "N"
     PRINT "Rispondere S o N"
     INPUT r$
LOOP


REM  Controllo di fine file
DO UNTIL EOF(1)
     LINE INPUT #1,Rec$
     PRINT Rec$
LOOP
```

GESTIONE DELLE PERIFERICHE

In questo capitolo verranno illustrate le istruzioni che riguardano l'input/output sulle unità periferiche di un sistema do elaborazione: *tastiera, video e stampante*.

La tastiera è la principale unità di input di un personal computer; con essa si possono immettere in memoria tutti i caratteri della tabella ASCII; il video e la stampante sono le unità di output.

• INPUT DA TASTIERA

L'immissione dei dati da tastiera si può effettuare con le seguenti istruzioni:

INPUT
LINE INPUT
INPUT$
INKEY$

- INPUT

Riceve i dati digitati e li memorizza in una o più variabili indicate in argomento.

Se vi sono più variabili debbono essere separate da virgole così come i corrispondenti dati digitati sulla tastiera.

I dati digitati debbono essere dello stesso tipo delle relative variabili (numero, stringa).

La richiesta di immissione viene indicata sul video con un punto interrogativo (?).

Esempi:

```
INPUT NOME$

INPUT PREZZO#, QTA%
```

Se si vuole esplicitare meglio del solo punto interrogativo la richiesta di immissione occorre esprimere, prima delle variabili, una costante a stringa con la frase di richiesta.

Esempi:

```
INPUT "Immetti il nominativo: ",nome$

INPUT "Rispondi Si o No ",Risposta$
```

- **LINE INPUT**

Questa istruzione viene utilizzata quando si deve immettere una intera riga di dati, contenente anche virgole e spazi, che la INPUT interpreta come delimitatori.

Si imposta con le stesse modalita' della INPUT

ma come argomento accetta soltanto una variabile stringa.

Esempio:

```
LINE INPUT "Immetti nome, indirizzo, telefono: ",Riga$
```

- **INPUT$**

E' una funzione che permette di immettere da tastietra un determinato numero di caratteri senza controllare l'immissione con il tasto *INVIO*.

Permette pertanto di memorizzare anche i tasti di controllo (*INVIO, ESC, tasti funzionali, BACKSPACE*).

Esempio:

```
REM  Memorizza 10 caratteri in Riga$
Riga$ = INPUT$(10)
```

- **INKEY$**

E' una funzione che controlla l'immissione di un carattere da tastiera senza interrompere l'esecuzione del programma (come invece avviene con le precedenti).

Se non viene premuto alcun tasto tra un

controllo di INKEY$ e il successivo, viene restituita una stringa vuota ("").

Esempio:

```
REM     Scrive i numeri fino a 100.000
REM     o finchè viene premuto un tasto.
DO
I=I+1
PRINT I
LOOP UNTIL I > 100.000 OR INKEY <> ""
```

- **OUTPUT SU VIDEO**

Lo schermo standard di un video è formato da 25 righe e 80 colonne; è quindi formato da 2000 celle, in ognuna delle quali si può visualizzare un carattere.

Per la gestione dei dati su video si utilizzano le seguenti istruzioni:

PRINT   Per visualizzare dati;

LOCATE   Per posizionare il cursore.

- PRINT

Esegue l'output sul video dei dati (costanti o variabili) indicati in argomento.

Se si visualizza più di una variabile o costante il separatore *virgola* (,) distanzia i dati utilizzando i tabulatori (blocchi di 14 colonne) dello schermo; il *punto e virgola* (;) indica di scrivere i dati a stringa uno di seguito all'altro senza spaziature.

I dati numerici sono sempre seguiti da uno spazio; se positivi hanno anche uno spazio iniziale (il segno *meno* se negativi).

Con la clausola USING è possibile formattare l'output fornendo una stringa di caratteri di controllo e riempimento che determina il formato e la lunghezza del dato visualizzato.

Per formattare un stringa si utilizzano i seguenti caratteri:

!	Visualizza solo il primo carattere;
\ \	Visualizza *n+2* caratteri (*n* è il numero di spazi tra le 2 barre rovesciate;
&	Visualizza l'intera stringa.

Per formattare un numero si utilizzano i seguenti caratteri:

#	Rappresenta la posizione di ciascuna cifra;
.	Indica il punto decimale e viene visualizzato solo se vi sono cifre decimali;
,	Posto a sinistra del punto decimale, fa stampare una virgola ogni tre cifre (divisione delle migliaia);
+	Posto all'inizio o alla fine della stringa di formato determina la visualizzazione del segno (all'inizio o alla fine del numero);
-	Posto alla fine della stringa di formato determina la visualizzazione del segno - alla fine di un numero negativo;
^^^^	Visualizza il numero in formato esponenziale;
$$	Visualizza un segno di $ prima del numero;
**	Riempie con asterischi gli spazi iniziali del campo numerico (protezione importi);

**$ Unisce gli effetti dei due simboli precedenti.

Esempi:

```
PRINT Nome$, Telefono$ 'Separa i
                'dati con la tabulazione

PRINT USING "###.##";A 'Visualizza
                'fino a 3 interi e 2
                'decimali;

PRINT USING "!"; A$ 'Visualizza il
                'primo carattere;

PRINT USING "+##.##"; A    'Visualiz-
                za fino a 3 interi e due
                'decimali e il segno.
```

- **LOCATE**

Con questa istruzione si può posizionare il cursore in qualunque punto dello schermo in modo che la successiva PRINT (o INPUT) inizi a visualizzare i dati da quel punto.

Si possono anche assegnare le dimensioni del cursore e la sua visibilità.

La posizione del cursore viene indicata dai primi 2 parametri che indicano il numero di riga (1-25) e di colonna (1-80).

Il terzo parametro indica la visibilità o meno del cursore (1=visibile, 0=spento).

Il quarto e quinto parametro indicano la forma e dimensione del cursore in righe di pixel (1-31).

Esempi:

```
LOCATE 6,10    'Posiziona il cursore a
               'riga 6 colonna 10;

LOCATE ,,0     'Spegne il cursore.
```

- ## OUTPUT SU STAMPANTE

Per indirizzare l'output su stampante si utilizza l'istruzione LPRINT che ha le stesse modalità della PRINT.

Poichè sulla stampante non è possibile utilizzare la LOCATE, nel controllo di colonna e riga si utilizzano le seguenti modalità:

- ### CONTROLLO DI COLONNA

Si effettua con la funzione TAB indicando

come argomento in parentesi il numero della colonna a cui iniziare a scrivere; se vi è una lista di dati da scrivere sulla stessa riga, ognuno di essi può essere preceduto da una funzione TAB.

Esempi:

```
LPRINT TAB(10); A$

LPRINT TAB(10); Nom$, TAB(40);
                      Ind$, TAB(70); Tel$.
```

- **CONTROLLO DI RIGA**

Per eseguire un salto riga basta scrivere l'istruzione LPRINT senza parametri.

Il salto pagina si esegue indicando come parametro di LPRINT il codice ASCII *12*, che invia il comando *Form Feed* (salto pagina) ed eventualmente il codice *13* che invia il comando *CR* (ritorno a capo).

Esempi:

```
REM    Salta riga
LPRINT

REM     Pagina nuova e a capo
LPRINT CHR$(12); CHR$(13)
```

GESTIONE DEI FILE SEQUENZIALI

Un file è un insieme di record e ogni record è composto da CAMPI, che normalmente sono ricorrrenti all'interno di ciascun record del file.

Nei file sequenziali i record sono registrati uno di seguito all'altro e l'accesso ad un certo record puo' avvenire soltanto scorrendo tutti quelli che lo precedono.

I record di un file sequenziale non possono essere modificati o cancellati; e' possibile soltanto aggiungere nuovi record in coda al file, ampliandolo.

Su un file sequenziale si possono eseguire le seguenti operazioni:

APERTURA DI UN FILE

CHIUSURA DI UN FILE

LETTURA DI UN RECORD

SCRITTURA DI UN RECORD

- ## APERTURA DI UN FILE

Qualunque operazione sui dati di un file e'

possibile soltanto dopo aver effettuato l'apertura, con l'istruzione OPEN, del file stesso, che puo' avvenire in diverse modalita' a seconda del tipo di operazioneche si vuole applicare ai dati (lettura, scrittura).

La OPEN assegna inoltre al file un numero (tra 1 e 255) che viene utilizzato dalle istruzioni di I/O come identificativo del file stesso, a cui resta associato fino a quando il file rimane aperto.

- **MODALITA' OUTPUT**

  Se il file specificato non esiste, viene creato e predisposto alla scrittura di un record.

  Se il file gia' esiste, vengono cancellati i dati preesistenti e viene predisposto alla scrittura di nuovi record.

  Esempio:

  ```
  OPEN "Anag.seq" for OUTPUT AS #1
  ```

- **MODALITA' APPEND**

  Se il file specificato non esiste, viene creato e predisposto alla scrittura di un record (come per OUTPUT).

Se il file gia' esiste, vengono mantenuti i record preesistenti e i nuovi vengono aggiunti in coda.

Esempio:

```
OPEN "Anag.seq" FOR APPEND AS #1
```

- **MODALITA' INPUT**

Si applica ad un file già esistente per predisporlo alla lettura di record.

Esempio:

```
OPEN "Anag.seq" FOR INPUT AS #1
```

- **CHIUSURA DI UN FILE**

La chiusura di un file si effettua con l'istruzione CLOSE che ha come argomento il numero associato al file dalla OPEN.

Se non si specifica l'argomento, vengono chiusi tutti i file aperti.

La chiusura di un file aperto in OUTPUT, determina la scrittura di eventuali record dati ancora presenti nel buffer di I/O del file e della configurazione di Fine File (EOF) e libera il

numero associato al file.

Esempi:

```
CLOSE       'Chiude tutti i file
CLOSE #1    'Chiude solo il file #1
```

- **SCRITTURA DI UN RECORD**

La scrittura di un record può essere fatta in aggiunta ai record già esistenti quando il file viene aperto in modalità APPEND, oppure in creazione quando il file viene aperto in modalità OUTPUT.

Con l'istruzione WRITE #*n* viene scritto un record composto dai campi indicati in argomento dopo il numero *n,* identificativo del file.

I campi vengono scritti separati dal carattere delimitatore *virgola* (,).

Esempio:

```
WRITE #1, Nome$, Indirizzo$, Telefono$
```

E' possibile scrivere un record anche con l'istruzione PRINT #*n* che non inserisce i caratteri delimitatori (*virgole* per separare i campi e *doppi apici* per racchiudere dati a stringa).

Esempi:

```
PRINT #1, Nome$, Indirizzo$, Telefono$
```

Per scriverlo con lo stesso formato della WRITE #:

```
PRINT #1, CHR$(34), Nome$, CHR$(34), ",",
          CHR$(34), Indir$, CHR$(34), ",",
          CHR$(34), Telef$, CHR$(34)
```

- **LETTURA DI UN RECORD**

La lettura dei dati di un file sequenziale è possibile solo dopo aver aperto il file in INPUT e avviene, record per record, secondo l'ordine di inserimento.

Si può controllare la fine del file tramite la funzione EOF (End Of File).

Con l'istruzione INPUT #*n* vengono letti i record individuando i singoli campi, che sono tra loro divisi dal carattere delimitatore *virgola* (,) e assegnandoli alle variabili indicate in argomento dopo il numero *n* identificativo del file.

Esempio:

```
IF EOF(1) THEN CLOSE: END
INPUT #1, nom$, qta%, pu#
```

Con l'istruzione LINE INPUT *#n* si legge un record intero (riga di dati) assegnandone il valore alla variabile a stringa indicata in argomento dopo il numero *n* identificativo del file.

Con questa istruzione è possibile leggere file di testo ASCII comprese le virgole.

Esempio:

```
OPEN "Testo1.seq" FOR INPUT AS #1
DO UNTIL EOF(1)
    LINE INPUT #1, Riga$
    PRINT Riga$
LOOP
CLOSE #1
END
```

GESTIONE DEI FILE RANDOM

L'organizzazione RANDOM di un file permette l'accesso ai singoli record in modo casuale, utilizzando come chiave di indirizzamento un numero intero positivo che indica la posizione fisica del record nell'ambito del file.

Per questo i record debbono avere tutti la stessa lunghezza (sono formati da campi di lunghezza prefissata).

Si dovrà predefinire una lunghezza per i campi a stringa e i campi numerici verranno registrati in un formato binario compresso che risparmia spazio disco e riduce i tempi di accesso: i numeri interi occupano 2 byte, quelli in semplice precisione 4 byte, quelli in doppia precisione 8 byte.

A differenza dei file sequenziali, in un file RANDOM è inoltre possibile la modifica di ogni singolo record mediante una operazione di riscrittura.

Con un file RANDOM si possono eseguire le seguenti operazioni:

APERTURA DI UN FILE E DEFINIZIONE DEI CAMPI DEL RECORD

CHIUSURA DI UN FILE

SCRITTURA DI UN RECORD

LETTURA DI UN RECORD

- **APERTURA DI UN FILE E DEFINIZIONE DEI CAMPI DEL RECORD**

Queste due operazioni sono sempre abbinate.

La OPEN abilita la gestione del file nell'unica modalità prevista RANDOM, che permette di effettuare sul file qualunque tipo di operazione.

Rispetto ai file sequenziali viene aggiunto il parametro LEN con cui si specifica la lunghezza del record.

Con l'istruzione FIELD si definisce la struttura del record indicando il nome e la lunghezza dei campi che lo compongono; i campi numerici saranno indicati come variabili a stringa della lunghezza indicata sopra.

Esempio:

```
OPEN "archiv" FOR RANDOM AS #1 LEN=40
FIELD #1, 30 AS nome$, 8 AS pu$,2 AS giac$
```

dove:

pu$ E' un valore numerico a doppia precisione convertito;

giac$ E' un valore numerico intero convertito.

Con questo sistema di definizione dei campi (FIELD) è necessario utilizzare appositi convertitori numerici sia in scrittura che in lettura, come vedremo nei capitoli successivi.

E' possibile eliminare queste funzioni di conversione utilizzando, per la definizione dei campi del record, la struttura TYPE...END TYPE invece della istruzione FIELD.

Dopo aver così definito il tipo di struttura occorre associargli una variabile con l'istruzione DIM.

Esempio:

Definiamo con TYPE l'esempio di record definito sopra con FIELD:

```
TYPE tiporec
    nome AS STRING*30
    pu AS DOUBLE
    giac AS INTEGER
END TYPE
DIM recmat AS tiporec
OPEN "archiv" FOR RANDOM AS #1
                LEN=LEN(recmat)
```

- **CHIUSURA DI UN FILE**

La chiusura di un file RANDOM avviene con le stesse modalità già viste per un file sequenziale.

- **SCRITTURA DI UN RECORD**

La registrazione di un record avviene con l'istruzione PUT che ha come primo parametro in argomento il numero del file, che dovrà essere stato aperto in modalità RANDOM.

Il secondo parametro indica la chiave di scrittura del record, un numero intero positivo che, se omesso, assume il valore di chiave, incrementato di 1 rispetto all'ultimo scritto, permettendo così di scrivere i record sequenzialmente.

Il terzo parametro è presente solo se il record è stato strutturato con la TYPE ed è il nome della variabile definita nella relativa DIM.

Esempio:

```
PUT #1, Chiave%, Recmat
```

Se il record è stato definito con l'istruzione FIELD è necessario allineare i dati nei campi definiti nella

FIELD con le istruzioni LSET o RSET (allineamento a sinistra o a destra) e, se numerici, convertirli in stringa con le funzioni di conversione MKI$, MKL$, MKS$, MKD$, rispettivamente per numeri interi, LONG, a semplice e doppia precisione.

Esempio:

```
LSET Nome$=N$
LSET Pu$=MKI$(Pu%)
LSET Giac$=MKD$(Giac#)
PUT #1, Chiave%
```

dove:

> N$, Pu%, Giac# sono i campi di memoria in cui risiedono i dati da memorizzare nel record e Nome$, Pu$, Giac$ i relativi campi definiti nella FIELD.

Se si vogliono aggiungere record ad un file Random già esistente, è necessario calcolare il valore iniziale della chiave in modo da non sovrascrivere i record già presenti.

Questo calcolo di prima chiave libera è possibile utilizzando la funzione LOF che fornisce la lunghezza in byte del file.

Poichè i file Random hanno i record a lunghezza fissa basta dividere la lunghezza del file per la

lunghezza del record, per ottenere il numero di chiave dell'ultimo record; incrementando di uno questo valore si ottiene il valore di chiave del primo record in estensione.

Esempio:

```
Chiave% = LOF(1) / LEN(Recmat) + 1
PUT #1, Chiave%, Recmat
```

Poichè nei file Random non avviene alcun controllo di chiave sui record già esistenti, l'operazione di riscrittura, necessaria per modificare un record, si effettua con l'istruzione PUT impostando la stessa chiave del record da modificare.

- **LETTURA DI UN RECORD**

La lettura di un record di un file Random avviene con l'istruzione GET che ha come primo parametro in argomento il numero del file che dovrà essere stato aperto in modalità RANDOM.

Il secondo parametro indica la chiave di lettura del record, un numero intero positivo, che, se omesso, permette di leggere il file sequenzialmente; se presente effettua un accesso casuale estraendo il record con chiave specificata.

Il terzo parametro è presente solo se il record è

stato strutturato con la TYPE ed è il nome della variabile definita nella relativa DIM.

Esempio:

```
GET #1, Chiave%, Recmat
```

Se il record è stato definito con l'istruzione FIELD è necessario, dopo la lettura, riconvertire i campi originariamente numerici utilizzando le funzioni di conversione CVI, CVL, CVS, CVD che effettuano l'operazione inversa a quella svolta da MKI$, MKL$, MKS$, MKD$.

Esempio:

```
GET #1
PRINT nome$, CVI(pu$), CVD(giac$)
```

Il controllo di fine file, quando si effettua una lettura sequenziale, o il controllo di validità della chiave, con l'accesso casuale, si può impostare utilizzando la funzione LOF con la tecnica vista in precedenza.

Esempio di lettura sequenziale:

```
nrec%=LOF(1) / LEN(recmat)
FOR chiave%=1 TO nrec%
GET #1, chiave%, recmat
      .
      .
```

```
NEXT
```

Esempio di lettura casuale:

```
nrec%=LOF(1) / LEN(recmat) + 1
DO
    INPUT "Immetti chiave di ricerca:", ch%
LOOP ch% > 0 AND ch% < nrec%
GET #1, ch%, recmat
```

MANIPOLAZIONE DELLE STRINGHE

In questo capitolo verranno trattate le principali istruzioni e funzioni che permettono la manipolazione dei dati in formato stringa.

Ricordiamo che una stringa è un campo alfanumerico che può contenere qualunque tipo di carattere rappresentato nella tabella dei codici ASCII; ogni carattere occupa un byte.

Una stringa si può esprimere sia in forma di *costante* che di *variabile*.

Una costante a stringa si esprime racchiudendola tra doppi apici (").

E' possibile esprimere un solo carattere anche con la funzione CHR$, indicando come argomento il valore del suo codice ASCII; questo modo è particolarmente utile quando si deve rappresentare un simbolo che non compare sulla tastiera.

Esempi:

```
PRINT "questa è una costante a stringa!"

PRINT chr$(156)     'visualizza il simbolo £
```

Una variabile a stringa può essere espressa nei

seguenti modi:

- Aggiungendo al nome il suffisso $ (nome$);

- Utilizzando l'istruzione DEFSTR che definisce una o più lettere come identificative di una variabile a stringa se usate come iniziale:

  DEFSTR A-F Indica che tutte le variabili il cui nome inizia con una lettera compresa in questo intervallo rappresentano una stringa.

- Dichiarando il nome della variabile con l'istruzione DIM...AS STRING:

  DIM Nome AS STRING   'stringa variabile

  Con questa istruzione è anche possibile dichiarare variabili stringa a lunghezza fissa:

  DIM Indirizzo AS STRING*30
  'stringa lunga 30 byte

Vediamo di seguito le principali operazioni possibili sulle stringhe.

- **CONCATENAMENTO**

E' possibile unire tra loro due o più stringhe utilizzando l'operatore +.

Esempio:

```
Nomin$=Cognome$+"  "+Nome$
PRINT Nomin$
```

- ## CONFRONTO

Il confronto tra stringhe avviene tramite i seguenti operatori di relazione:

=	uguale;
<	minore;
>	maggiore;
<>	diverso;
<=	minore o uguale;
>=	maggiore o uguale.

Il confronto avviene carattere per carattere partendo da sinistra e prosegue fino a quando non incontra due caratteri diversi (o fino alla fine in caso di uguaglianza).

Il risultato di un confronto è determinato in base al valore dei caratteri nella codifica ASCII per cui le lettere maiuscole hanno un valore minore delle

minuscole e i numeri precedono le lettere.

Occorre ricordare che anche lo spazio è un simbolo che ha un valore nella codifica ASCII, per cui, il confronto tra due stringhe, di lunghezza diversa non potrà mai dare uguaglianza in quanto vengono considerati anche eventuali spazi finali.

Esempio:

```
A$ = "ROSSI"            'Il risultato
B$ = "ROSSI    "        'del confronto
IF A$ = B$ THEN...      'sara' diverso
```

- **RICERCA**

E' possibile ricercare all'interno di una stringa una precisa sequenza di caratteri (sottostringa) con la funzione INSTR che fornisce la posizione della prima occorrenza della sottostringa all'interno della stringa.

Se non viene rilevata alcuna corrispondenza fornisce il valore 0.

Esempio:

```
PRINT "posizione della parola BASIC=";
                    INSTR (riga$, "Basic")
```

- ## FUNZIONI DI STRINGA

Esistono in Basic diverse funzioni che operano sulle stringhe; possiamo raggrupparle nelle seguenti tipologie:

- ### ESTRAZIONE DI CARATTERI

Con le funzioni LEFT$, RIGHT$ e MID$ è possibile estrarre caratteri dalla stringa, indicata come primo parametro in argomento, rispettivamente da sinistra, da destra e dalla parte centrale.

Il secondo parametro indica il numero di caratteri da estrarre per le prime due funzioni, mentre la funzione MID$ ha come secondo parametro la posizione del primo carattere da estrarre e come terzo parametro il numero di caratteri da estrarre.

Esempio:

```
a$ = "VIVACIZZARE IL CORSO"
b$ = "UTILIZZANDO"
c$ = "QUICK-BASIC"
PRINT LEFT$(a$,4); " "; MID$(b$,3,2);
                        " "; RIGHT$(c$,5)
REM       scrive VIVA IL BASIC
```

Con le funzioni LTRIM$ e RTRIM$ si eliminano rispettivamente gli spazi iniziali e finali della stringa indicata in argomento.

Esempi:

```
PRINT LTRIM$("     ROSSI")
            'scrive "ROSSI"

PRINT RTRIM$("ROSSI     ")
            'scrive "ROSSI"
```

Con quest'ultima funzione è possibile anche effettuare correttamente il confronto tra due stringhe di lunghezza diversa eliminando gli eventuali spazi finali in quella più lunga.

Esempio:

```
a$ = "ROSSI"              'Stringa di 5 car.
b$ = "ROSSI   "           'Stringa di 8 car.
if a$ = b$ THEN...        'Restituisce
                              'diverso
if a$ = rtrim$(b$) THEN...    'Restituisce
                              'uguale
```

- **STRINGHE DI CARATTER**

La funzione STRING$ fornisce una stringa della lunghezza indicata dal primo parametro in argomento, composta da uno

stesso carattere indicato come secondo parametro.

Esempio:

```
PRINT STRING$(80,"*")   'Scrive una
                        'riga di 80 asterischi.
```

La funzione SPACE$ fornisce una stringa di spazi della lunghezza indicata in argomento.

- **SOSTITUZIONE DI MAIUSCOLE E MINUSCOLE**

Con le funzioni UCASE$ e LCASE$ è possibile trasformare rispettivamente le lettere minuscole della stringa in argomento in maiuscole e viceversa.

Sono molto utili per effettuare estrazioni o confronti di stringhe senza doversi preoccupare di come i dati sono stati immessi.

Esempio:

```
a$= "ROSSI"
b$ = "Rossi"
if a$ = b$ THEN...              'diverso
if a$ = ucase$(b$) THEN...      'uguale
```

- **LUNGHEZZA DI UNA STRINGA**

Con la funzione LEN si può ottenere la lunghezza della stringa indicata in argomento.

Esempi:

```
A$ = "BASIC"
PRINT LEN(A$)          'Scrive 5

REM   Centra automaticamente il titolo
LOCATE 2, INT(40 - LEN(titolo$) / 2)
PRINT TITOLO$
```

- **NUMERI E STRINGHE**

Le funzioni STR$ e VAL trasformano un numero in stringa e viceversa.

Permettono di superare l'incompatibilità, nelle assegnazioni e nei confronti, tra campi numerici e stringa.

Esempi:

```
a%=VAL(scelta$)

PRINT STR$ (i)
```

OPERAZIONI MATEMATICHE

In Basic le operazioni matematiche si eseguono con un formalismo simile a quello utilizzato nell'impostazione delle espressioni algebriche.

Si possono utilizzare i seguenti operatori e simboli:

=	segno di uguale;
+	operatore di somma;
-	operatore di sottrazione;
*	operatore di moltiplicazione;
/	operatore di divisione;
^	elevazione a potenza;
( )	parentesi.

Il primo elemento dovrà essere la variabile in cui verrà memorizzato il risultato, seguito dal simbolo di uguale; segue l'espressione che può essere composta da variabili e costanti numeriche, parentesi, operatori e funzioni.

Queste ultime verranno illustrate nei capitoli

successivi.

Il simbolo - può anche essere utilizzato come segno algebrico (negativo).

Esempi:

```
A=B * H / 2

X = (A - B) / (C + D) ^ 2

C = C + 1       'Contatore

T = T + A       'Totalizzatore

X = A / - B     'Uso del - come segno algebrico.
```

- **FUNZIONI TRIGONOMETRICHE**

  Le funzioni trigonometriche *Seno*, *Coseno*, *Tangente* e *Arcotangente* si esprimono con gli operatori SIN, COS, TAN, ATN.

  L'argomento in parentesi è un'espressione numerica che, per i primi tre operatori, indica la misura di un angolo espresso in radianti, per ATN indica la misura di una tangente di cui si vuole l'angolo (espresso in radianti).

  Si ricorda che per convertire la misura di un angolo da gradi a radianti si moltiplicano i gradi

per *pigreco*/180, dove *pigreco*=3.141593.

Esempio:

```
x = COS(75 * (3,141593 / 180))
REM   Fornisce il cos di 75°
```

- **FUNZIONI LOGARITMICHE**

La funzione LOG fornisce il logaritmo naturale (a base *e* con *e* = 2,718282) dell'espressione numerica indicata in argomento, che deve essere positiva.

Si ricorda che per ottenere il logaritmo decimale (a base ***10***) basta dividere il logaritmo naturale per il logaritmo naturale di 10.

Esempi:

```
x = LOG(y)              'Logaritmo naturale

x = LOG(y) / log(10)    'Logaritmo decimale
```

Si può indicare in questo capitolo anche la funzione EXP (*esponenziale*), che esegue l'operazione inversa al logaritmo naturale: restituisce il risultato dell'elevazione a potenza di *e* (base dei logaritmi naturali) per il numero espresso in argomento.

Esempio:

```
X = EXP(Y)    E' come scrivere x=2,718282^y
```

- **FUNZIONI MATEMATICHE**

Il Basic mette a disposizione altre funzioni matematiche che indichiamo di seguito:

- **SQR** Fornisce la radice quadrata (Square Root) del numero espresso in argomento, che deve essere positivo.

  Esempio:

  ```
  X = SQR(Y)
  ```

- **ABS** Fornisce il valore assoluto del numero espresso in argomento.

  Esempio:

  ```
  x = ABS(-5)        'Fornisce 5
  ```

- **INT** Fornisce la sola parte intera del numero espresso in argomento.

  Esempio:

  ```
  IF x = INT(x) THEN... 'Se la condizione è
  ```

```
'vera x è intero, se è falsa è decimale.
```

- **RND** fornisce un numero a caso, in semplice precisione compreso tra 0 e 1.

  Esempio:

  ```
  PRINT INT(RND * 6) + 1   'Simula il tiro
                           'di un dado.
  ```

- **SGN** fornisce il segno algebrico del numero espresso in argomento (0 se il numero è nullo, 1 se è positivo, -1 se è negativo).

  Esempio:

  ```
  IF SGN(X) = -1 THEN
      PRINT "il numero è negativo"
  END IF
  ```

LA GRAFICA

In questo capitolo verranno sinteticamente trattate le principali istruzioni e funzioni grafiche che il Basic mette a disposizione e che permettono, grazie al colore e alla elevata risoluzione grafica dello schermo di un moderno Personal Computer, di abbellire i programmi tradizionali con funzioni visive particolarmente gradevoli e di produrre programmi grafici di tipo scientifico, educativo e di evasione (giochi).

Si possono infatti ottenere le seguenti funzionalità:

- *Predisposizione dello schermo;*
- *Gestione dei colori;*
- *Disegno di linee e riquadri;*
- *Disegno di cerchi, ellissi, spicchi e archi;*
- *Utilizzo di finestre;*
- *Animazione di immagini.*

Un computer, per poter eseguire programmi grafici, deve essere fornito di scheda grafica a colori (CGA, EGA, VGA) e relativo monitor grafico a PIXEL.

I Pixel sono dei punti con cui si formano le immagini

sullo schermo, accendendoli, spegnendoli e attribuendo loro un determinato colore.

Il numero di Pixel di cui è composto lo schermo determina la risoluzione grafica delle immagini rappresentate: più è grande il loro numero, più alta è la risoluzione grafica, cioè la nitidezza dell'output.

E' possibile scegliere tra diverse risoluzioni grafiche tramite l'istruzione SCREEN: ad esempio SCREEN 1 fornisce una risoluzione di 320 Pixel orizzontali per 200 verticali, SCREEN 2 fornisce una risoluzione di 640 x 200 Pixel.

Risoluzioni più alte sono possibili in funzione del tipo di scheda grafica utilizzata.

Ogni Pixel è individuato da 2 coordinate che ne determinano la posizione sullo schermo e sono indicate in parentesi, separate dalla virgola; la prima è la coordinata di *colonna*, la seconda quella di *riga*; il punto (0,0) è l'angolo in alto a sinistra.

- **PREDISPOSIZIONE DELLO SCHERMO**

  Prima di impostare istruzioni o funzioni grafiche è necessario predisporre lo schermo nella modalità grafica desiderata: numero di Pixel che lo compongono, definizione dei colori disponibili e della memoria video da visualizzare.

Queste caratteristiche vengono fornite dall'istruzione SCREEN:

- il primo parametro indica la risoluzione grafica (numero di Pixel) e puo' essere un numero da 1 a 13 (per conoscere le caratteristiche di ogni valore si consulti la guida in linea alla voce *Modalita' schermo*);

- il secondo parametro indica se si utilizza lo schermo con visualizzazione monocromatica o a colori (in modalità schermo 0 o 1);

- il terzo e quarto parametro indicano la parte di memoria (pagina di schermo) attiva e visualizzata.

E' importante ricordare che questa istruzione deve sempre precedere qualunque altra istruzione grafica e che le modalità schermo che imposta sono subordinate all'Hardware (adattatore grafico e monitor) installato.

Esempio:

```
SCREEN 2      'Risoluzione grafica 640 x 200
```

- **GESTIONE DEI COLORI**

Nella gestione di un output grafico si possono utilizzare diversi colori sia per lo sfondo che per il

primo piano.

Con modalità schermo 1 e 2 i colori di sfondo sono 16 e quelli di primo piano sono definibili in gruppi di 4 (PALETTE).

Le caratteristiche di colore si impostano con la istruzione COLOR, con le modalità indicate nel *Sommario.*

Nelle varie istruzioni grafiche che vedremo di seguito si può indicare un parametro *colore*, che può essere utilizzato in alternativa al colore di primo piano impostato dall'istruzione COLOR o per scegliere il colore della PALETTE impostata dall'istruzione COLOR.

Esempio:

```
SCREEN 1
COLOR 10, 0  'Sfondo verde chiaro
             'Palette=verde, rosso, marrone
```

Con l'istruzione PALETTE è possibile modificare i colori standard assegnando ad ogni attributo (numero di colore di primo piano) qualunque colore.

Mediante l'opzione USING è possibile modificare contemporaneamente tutti i colori assegnati ad ogni attributo.

Gli attributi e i colori disponibili dipendono dal tipo di scheda grafica del sistema e dalla modalità schermo impostata dall'ultima istruzione SCREEN.

Per ulteriori informazioni sugli argomenti trattati sopra, si consiglia di consultare la guida in linea alle voci *Attributi e valori di colore* e *Modalità schermo*.

- **DISEGNO DI LINEE E RIQUADRI**

Per tracciare una linea si utilizza l'istruzione LINE, specificando le coordinate delle due estremità del segmento.

Se le coordinate sono precedute dall'opzione STEP, invece di indicare i valori assoluti a partire dall'angolo superiore sinistro con coordinate (0, 0), si assumono relative all'ultimo punto tracciato sullo schermo (posizione corrente del cursore grafico).

Esempio:

```
REM   Disegna una scala
SCREEN 2
LINE (10, 10) - (30, 10)    'primo tratto orizz.
FOR x% = 1 to 15
    LINE -STEP(0, 8)        'tratto verticale
    LINE -STEP(30, 0)       'tratto orizzontale
NEXT
```

Il parametro che segue le coordinate, separato da virgola, indica il colore del segmento ed è un attributo il cui valore dipende dai parametri di predisposizione dello schermo.

Esempio:

```
SCREEN 1
COLOR 2,0          'Sfondo verde, palette 0
LINE (10, 10) - (60, 60), 2      'Rosso
```

Per disegnare un riquadro basta aggiungere, dopo l'opzione colore, il parametro B (Box); le coordinate dovranno indicare rispettivamente l'angolo superiore sinistro e quello inferiore destro del riquadro.
Per ottenere un riquadro pieno si utilizza il parametro BF.

Esempio:

```
SCREEN 1
REM   Riquadro
LINE (10, 10) - (120, 40),, B
REM   Riquadro pieno
LINE STEP (10, 10) - STEP (40, 40),, BF
```

- **DISEGNO DI CERCHI, ELLISSI, SPICCHI E ARCHI**

Tutte queste figure vengono disegnate con

l'istruzione CIRCLE.

Per disegnare un cerchio basta fornire le coordinate del centro (assolute o relative) come primo parametro e la misura del raggio come secondo; il terzo parametro, facoltativo, indica l'attributo di colore.

Esempio:

```
SCREEN 2
CIRCLE (250, 100)
```

Per disegnare una ellisse occorre fornire, oltre ai parametri già visti per il cerchio, un parametro che indica il rapporto tra i due raggi:

> se il rapporto è maggiore di 1, l'ellisse si estenderà verticalmente,
>
> se è minore di 1, l'ellisse si estendera' orizzontalmente.

Il secondo parametro (misura del raggio) indica le dimensioni del raggio maggiore.

Poichè il rapporto tra i due raggi è l'ultimo parametro dell'istruzione CIRCLE, dovrà essere preceduto da 4 virgole (o 3 se si indica l'attributo colore).

Esempio:

```
SCREEN 1
CIRCLE (80, 90), 75,,,, 3        Ellisse verticale
CIRCLE (220, 90), 75, 2,,, 3/10    Ellisse
                              'orizzontale magenta
```

Per disegnare un arco (di cerchio o di ellisse) occorre aggiungere ai parametri di definizione che abbiamo visto sopra, altri due parametri che indicano gli angoli, espressi in radianti, di inizio e fine dell'arco.

L'unità di misura dei radianti è *Pigreco* (3,141593) che corrisponde a 180 gradi.

I parametri di inizio e fine dell'arco sono espressi in quarta e quinta posizione.

Esempio:

```
REM   Arco di cechio da 90 a 720 gradi
SCREEN 1
CIRCLE (80, 90), 50,, 3.14 / 2, 3/2 * 3.14
```

Per disegnare uno spicchio (grafici a torta) è sufficiente definire il relativo arco di cerchio, con i parametri di inizio e fine dell'arco negativi.

in questo modo verranno disegnati anche i due segmenti di retta che uniscono gli estremi dell'arco al centro, a formare uno spicchio o una torta priva

di uno spicchio a seconda della dimensione dell'arco.

Esempio:

```
SCREEN 2
pg=3.14159
REM       Torta
CIRCLE (250, 100), 100,, -pg, -pg * 2 / 3
REM       Spicchio
CIRCLE (200, 130), 100,, -pg * 2 / 3, -pg
```

- **UTILIZZO DI FINESTRE**

E' possibile utilizzare finestre di varie dimensioni in cui inserire gli oggetti grafici, invece di utilizzare l'intero schermo.

Una finestra si definisce con l'istruzione VIEW, indicando come primo parametro le coordinate degli angoli della finestra (come per i riquadri), il secondo e il terzo parametro (opzionali) indicano il colore di fondo e del bordo della finestra.

Esempio:

```
REM   Finestra con sfondo azzurro e
REM                    bordo magenta
SCREEN 1
VIEW (50, 60) - (175, 150), 1, 2
```

L'indirizzamento (coordinate di Pixel), con questa definizione delle finestre, è di tipo relativo, in modo che l'angolo della finestra in alto a sinistra ha coordinate (0,0).

E' possibile mantenere all'interno di una finestra l'indirizzamento assoluto (riferito cioè alle coordinate dello schermo) con l'opzione SCREEN.

Esempio:

```
SCREEN 1
VIEW SCREEN (50, 60) - (175, 150), 1, 2
```

Con questa rappresentazione, l'output grafico con coordinate esterne a quelle di definizione della finestra non appare sullo schermo.

E' possibile ridefinire la scala di coordinate di una finestra (o dell'intero schermo) con l'istruzione WINDOW.

La nuova scala così definita prende il nome di scala a coordinate logiche; si definiscono invece coordinate fisiche quelle originali a PIXEL, che è possibile ripristinare con un'istruzione WINDOW priva di argomenti.

Esempio:

```
SCREEN 1
VIEW (50, 60) - (170, 150), 1, 2
```

```
WINDOW (0, 0) - (100, 75)
```

In questo esempio l'istruzione WINDOW definisce per la finestra una scala da 0 a 100 per l'*ascissa* (asse x) e da 0 a 75 per l'*ordinata* (asse y) con origine delle coordinate nell'angolo in basso a sinistra.

Se si vuole mantenere l'origine delle coordinate nell'angolo in alto a sinistra della finestra si deve utilizzare l'opzione SCREEN.

Esempio:

```
SCREEN 1
VIEW (50, 60) - (170, 150), 1, 2
WINDOW SCREEN (0, 0) - (100, 75)
```

In questo esempio l'istruzione WINDOW SCREEN definisce per la finestra una scala simile a quella dell'esempio precedente ma con origine delle coordinate nell'angolo in alto a sinistra.

- **ISTRUZIONE DRAW**

E' una macro istruzione particolarmente potente che ha come parametro una stringa contenente la sequenza dei comandi grafici abbreviati che verranno eseguiti.

Si possono disegnare, ruotare, ridimensionare e

colorare oggetti.

Consultando la guida in linea alla voce *DRAW* si può visualizzare il dettaglio dei comandi che si possono utilizzare.

Esempio:

```
SCREEN 1
DRAW "F60 L120 E60"
DRAW "BD30 P1,2 C3 M30, -30"
```

- **ANIMAZIONE DI IMMAGINI**

Possiamo indicare due tecniche per animare oggetti disegnati.

Con la prima è possibile utilizzando soltanto le istruzioni per disegnare viste in precedenza e operando nel modo seguente:

> Si disegni, ad esempio, un cerchio con l'istruzione CIRCLE, poi si cancelli ridisegnandolo con il colore di sfondo e si disegni nuovamente il cerchio con il centro spostato e così via.
>
> Questa tecnica permette di animare oggetti semplici e il movimento è piuttosto lento.

Per avere un'animazione più veloce con oggetti

anche complessi si possono utilizzare le istruzioni grafiche GET e PUT.

La GET preleva dallo schermo un'immagine e la copia in memoria; l'immagine viene individuata dalle coordinate degli angoli diametralmente opposti di un riquadro che la racchiude.

Con la PUT è possibile visualizzare un'immagine precedentemente memorizzata con GET, in un punto qualunque dello schermo individuato dalle coordinate di un unico punto corrispondente all'angolo in alto a sinistra del riquadro da copiare.

L'interazione dell'immagine copiata con gli oggetti già presenti sullo schermo viene effettuata con una delle seguenti opzioni da inserire come ultimo parametro della PUT:

AND Unisce l'immagine memorizzata con una precedente;

OR Sovrappone un'immagine memorizzata a quella già esistente;

PSET Disegna un'immagine memorizzata cancellando una immagine già esistente.

PRESET Disegna un'immagine memorizzata con inversione dei colori cancellando l'immagine esi-

stente.

XOR Disegna un'immagine memorizzata o cancella un'immagine disegnata in precedenza conservando allo stesso tempo lo sfondo.
Ciò produce degli effetti di animazione.

Esempio:

```
SCREEN 1
DIM C%(1 TO 200)
X1% = 0: X2% = 10: Y1% = 0: Y2% = 10
LINE (X1%, Y1%) - (X2%, Y2%), 2, BF
GET (X1%, Y1%) - (X2%, Y2%), C%
DO
PUT (X1%, Y1%), C%, XOR
X1% = RND * 300
Y1% = RND * 180
PUT (X1%, Y1%), C%
LOOP WHILE INKEY$ = ""
```

SOMMARIO

DELLE

ISTRUZIONI E FUNZIONI

SOMMARIO DELLE ISTRUZIONI E FUNZIONI

Vengono di seguito descritte le principali istruzioni e funzioni, in ordine alfabetico per una più facile ricerca.

Nella sintassi delle istruzioni e funzioni del Basic si assumono le seguenti convenzioni:

- Le voci scritte in **MAIUSCOLO** sono parole riservate del Basic e debbono essere digitate esattamente come scritte nel manuale;

- I parametri formali sono scritti in *minuscolo corsivo*; al loro posto occorre fornire la corretta informazione;

- Le parentesi quadre ([ ]) indicano voci o parametri facoltativi;

- I puntini sospensivi (...) indicano la possibilita' di ripetizione dell'argomento precedente;

- La barra verticale (|) separa elementi tra loro alternativi (condizione OR).

Negi esempi i nomi delle variabili saranno scritti indifferentemente in maiuscolo o minuscolo, in accordo con le regole del Basic.

ABS

Fornisce il valore assoluto di una espressione numerica.

TIPO Funzione matematica

SINTASSI

ABS(*espressione-numerica*)

ESEMPI

```
A = ABS(B)

X = ABS(X1 - X2)
```

ASC

Fornisce il valore numerico corrispondente al codice ASCII del primo carattere di una espressione a stringa.

TIPO Funzione

SINTASSI

ASC(*espressione-stringa*)

ESEMPI

```
a = ASC(nome$)

n =ASC(MID$(nome$,i,1))
```

ATN

Fornisce l'arcotangente di una espressione numerica.

TIPO Funzione matematica

SINTASSI

ATN*(espressione-numerica)*

ESEMPIO

```
A = ATN(B#)
```

BEEP

Fa emettere un suono all'altoparlante.

TIPO Istruzione

SINTASSI

BEEP

BLOAD

Carica in memoria un file di immagine della memoria, generato in precedenza con BSAVE.

TIPO Istruzione

SINTASSI

BLOAD *spec-file [,offset]*

dove:

spec-file	espressione a stringa che specifica il file e l'unità di input da cui viene caricato;
offset	espressione numerica che indica l'indirizzo di memoria in cui caricare il file; se omesso viene assunto l'indirizzo specificato in BSAVE.

ESEMPIO

```
BLOAD "A:grafico.grh"
```

BSAVE

Trasferisce sul file specificato il contenuto di un'area di memoria.

TIPO Istruzione

SINTASSI

BSAVE *specfile, offset, lunghezza*

dove:

spec-file espressione a stringa che specifica il file e l'unità di output in cui memorizzare;

offset espressione numerica che indica l'indirizzo iniziale dell'area di memoria da trasferire su file;

lunghezza dimensione, in numero di byte, dell'area da trasferire.

ESEMPIO

```
BSAVE "A:grafico.grh",VARPTR(graf(1)),2500
```

CALL

Trasferisce il controllo al sottoprogramma Basic indicato (definito dall'istruzione SUB).

TIPO Istruzione

SINTASSI

CALL *nome[(argomenti)]*

dove:

nome	della SUB del Basic a cui passare il controllo;
argomenti	elenco delle variabili o costanti da passare al sottoprogramma; i vari argomenti sono separati da virgole.

ESEMPIO

```
CALL contr-data(giorno,mese,anno)
```

CALLS

Trasferisce il controllo ad un modulo scritto in un altro linguaggio.

TIPO Istruzione

SINTASSI

CALLS *nome [(argomenti)]*

dove:

i parametri hanno la stessa funzionalità dell'istruzione CALL.

ESEMPIO

CALLS

CALL ABSOLUTE

Trasferisce il controllo ad un modulo in linguaggio macchina.

TIPO Istruzione

SINTASSI

CALL ABSOLUTE *([argomenti] variabile-intera)*

dove:

argomenti nomi degli argomenti da passare;

variabile-intera contiene l'offset dal segmento di codice cerrente, impostato con DEF SEG all'indirizzo iniziale del modulo chiamato.

ESEMPIO

```
CALL ABSOLUTE (a, b, VARPTR(absprog))
```

CALL INTERRUPT

Consente, da un programma Basic, di eseguire chiamate di sistema Dos

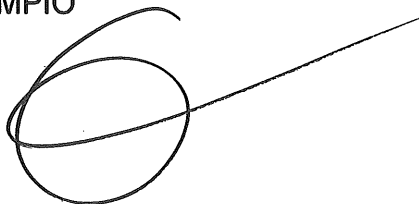
TIPO Istruzione

SINTASSI

CALL INTERRUPT *(num-int, reg-prima, reg-dopo)*

dove:

num-int	numero di interrupt del Dos (intero tra 0 e 255);
reg-prima	variabile che contiene i valori dei registri utilizzati all'esecuzione dell'interrupt;
reg-dopo	variabile che contiene i valori dei registri dopo l'esecuzione dell'interrupt.

ESEMPIO

CDBL

Converte una espressione numerica in un numero in doppia precisione.

TIPO Funzione matematica

SINTASSI

CDBL(*espressione-numerica*)

ESEMPIO

```
X# = CDBL(A/3)
```

CHAIN

Trasferisce il controllo al programma specificato.

TIPO Istruzione

SINTASSI

CHAIN *spec-file*

dove:

spec-file	espressione a stringa che definisce il programma a cui passare il controllo; può contenere l'indicazione DRIVE:PATH.

ESEMPIO

```
CHAIN A:\BASIC\PROGR1.BAS
```

CHDIR

Cambia la directory corrente nell'unità disco specificata (comando Dos).

TIPO Istruzione Dos

SINTASSI

CHDIR *"[drive:]path"*

dove:

[drive:]path identifica la directory che diventerà predefinita.

ESEMPIO

```
CHDIR "C:\BASIC\Arch-01"
```

CHR$

Fornisce il carattere il cui codice ASCII corrisponde all'argomento.

TIPO Funzione

SINTASSI

CHR$(*codice*)

dove:

codice espressione numerica il cui valore indica il codice di un carattere ASCII.

ESEMPIO

```
PRINT CHR$(12);CHR$(13)    'comando di
                           'pagina nuova e a capo
```

CINT

Converte una espressione numerica in un numero intero (2 byte binari) arrotondando la parte decimale.
Il valore rappresentabile è compreso tra -32.768 e +32.767.

TIPO Funzione

SINTASSI

CINT(*espressione-numerica*)

ESEMPIO

```
PRINT CINT(A)
```

CIRCLE

Disegna un cerchio o una ellisse specificando raggio e centro.

TIPO Istruzione grafica

SINTASSI

CIRCLE *[STEP](x,y),raggio[,[colore] [,[inizio][,[fine][,dimensione]]]]*

dove:

STEP	indica che le coordinate sono relative alla posizione corrente del cursore;
x,y	coordinate del centro;
raggio	dimensione del raggio espresso nell'unità del sistema di coordinate corrente;
colore	attributo di colore;
inizio	angolo d'inizio dell'arco espresso in radianti;
fine	angolo di fine dell'arco espresso in radianti;

dimensione rapporto tra gli assi *x* e *y* usato per definire una ellisse.

ESEMPI

```
CIRCLE (320,100),200

CIRCLE STEP (0,0),100

CIRCLE STEP (100,150),94,,0,0,6.28
```

CLEAR

Reindirizza tutte le variabili del programma, chiude i file e inizializza lo stack.

TIPO Istruzione

SINTASSI

CLEAR*[,,stack]*

dove:

stack dimensione in byte dello stack.

ESEMPIO

CLEAR,,2000

CLNG

Converte una espressione numerica in un numero INTERO LONG (4 byte binari), arrotondando la parte decimale.
Il valore rappresentabile è compreso tra
-2.147.483.648 e +2.147.483.647.

TIPO Funzione

SINTASSI

CLNG(*espressione-numerica*)

ESEMPIO

```
PRINT CLNG(A#)
```

CLOSE

Chiude uno o più file.

TIPO Istruzione

SINTASSI

CLOSE *[num-file][,...]*

dove:

num-file è un numero con cui è stato aperto il file.
Se non viene indicato alcun argomento vengono chiusi tutti i file aperti.

ESEMPI

CLOSE #1,#2

CLOSE 1

CLOSE

CLS

Pulisce lo schermo (comando Dos).

TIPO Istruzione Dos

SINTASSI

CLS *[0|1|2]*

dove:

0	cancella qualsiasi testo o grafico;
1	cancella solo la finestra grafica attiva;
2	cancella solo la finestra di testo lasciando inalterata l'ultima riga dello schermo.

Omettendo l'argomento viene cancellata la finestra grafica o quella di testo, a seconda della precedente istruzione VIEW.

ESEMPI

```
CLS

CLS 0
```

COLOR

Imposta i colori per la visualizzazione.

TIPO Istruzione grafica

SINTASSI

COLOR *[primo-piano][,[sfondo][,bordo]]* modalità 0

COLOR *[sfondo][,palette]* modalità 1-7

COLOR *[primo-piano][,sfondo* modalità 7-11

COLOR *[primo-piano* modalità 12-13

dove:

primo-piano numero che identifica il colore di primo piano dello schermo;

sfondo numero che identifica il colore di sfondo dello schermo;

bordo numero che identifica il colore di contorno dello schermo;

palette un numero (0-1) che identifica un insieme di tre colori:

0 = verde, rosso, marrone;

1 = azzurro, magenta, bianco.

Di seguito è riportata la tabella dei colori:

0	nero	8	grigio
1	blu	9	blu chiaro
2	verde	10	verde chiaro
3	azzurro	11	azzurro chiaro
4	rosso	12	rosso chiaro
5	magenta	13	magenta chiaro
6	marrone	14	giallo
7	bianco	15	bianco

I valori di colore indicati si riferiscono alla modalità 1.

La modalità schermo utilizzabile dipende dall'adattatore grafico del sistema e viene impostata dall'istruzione SCREEN.

ESEMPI

```
SCREEN 0
COLOR 1,2,3

SCREEN 1
COLOR 2,1
```

COM

Attiva, disattiva o sospende la gestione degli errori relativi alla porta di comunicazione specificata.

TIPO Istruzione

SINTASSI

COM*(n) ON|OFF|STOP*

dove:

n numero della porta seriale (1 o 2)

ESEMPIO

```
COM(1) ON
```

COMMAND$

Ripristina la riga di comando Dos utilizzata per richiamare il programma.

TIPO Funzione

SINTASSI

COMMAND$

ESEMPIO

COMMAND$

COMMON

Definisce le variabili globali che possono essere condivise in un programma o tra più programmi collegati.

TIPO Istruzione

SINTASSI

COMMON *[SHARED] [elenco-variabili]*

dove:

SHARED indica che le variabili sono condivise con le procedure SUB o FUNCTION;

elenco-variabili elenco delle variabili da condividere; si esprimono con la seguente sintassi:

variabile AS *tipo*,...;

tipo indica il tipo della variabile e può essere: INTEGER, LONG, SINGLE, DOUBLE o STRING.

ESEMPIO

```
COMMON A AS STRING,B AS SINGLE
```

CONST

Dichiara le costanti simboliche da utilizzare al posto di valori numerici o di stringa.

TIPO Istruzione

SINTASSI

CONST *nome-costante=espressione,...*

dove:

nome-costante nome da assegnare alla costante;

espressione costante o espressione di costanti e operatori aritmetici o logici (escluso l'elevamento a potenza e le funzioni intrinseche).

ESEMPIO

```
CONST  PI = 3.141.593
       FINE = -1
```

COS

Fornisce il coseno di un angolo espresso in radianti.

TIPO Funzione matematica

SINTASSI

COS(*espressione - numerica*)

dove:

espressione - numerica è un valore o una espressione di qualsiasi tipo numerico.
Per convertire i gradi in radianti, moltiplicare i gradi per (Pgreco/180)

ESEMPIO

```
X = COS(120 * (Pgreco/180))
```

CSNG

Converte una espressione numerica in un numero in semplice precisione.

TIPO Funzione matematica

SINTASSI

CSNG(*espressione - numerica*)

ESEMPIO

X = CSNG(1/3)

CSRLIN

Fornisce la posizione di riga attuale del cursore

TIPO Funzione

SINTASSI

CSRLIN

ESEMPIO

CSRLIN

CVI, CVL, CVS, CVD

Convertono in numeri stringhe contenenti valori numerici.
Vengono utilizzate con le istruzioni FIELD e GET per leggere valori numerici da un file RANDOM.

TIPO Funzione

SINTASSI

CVI(*stringa-2*) converte in numero intero;

CVL(*stringa-4*) converte in numero intero LONG;

CVS(*stringa-4*) converte in numero a semplice precisione;

CVD(*stringa-8)* converte in numero a doppia precisione.

dove:

stringa-2 stringa di 2 byte;

stringa-4 stringa di 4 byte;

stringa-8 stringa di 8 byte;

NOTA

Queste funzioni sono collegate alle corrispondenti funzioni per convertire un numero in stringa MKI$, MKL$, MKS$, MKD$.

ESEMPI

```
PRINT CVI(A$)

PRINT CVD(ND$)
```

CVDMBF, CVSMBS

Convertono stringhe che contengono numeri in formato binario Microsoft in numeri in formato IEEE. Vengono utilizzate con le istruzioni FIELD e GET per i file RANDOM.

TIPO Funzione

SINTASSI

CVSMBF(*stringa-4*) converte in numero a semplice precisione;

CVDMBF(*stringa-8*) converte in numero a doppia precisione;

dove:

stringa-4 stringa di 4 byte contenente un numero in binario;

stringa-8 stringa di 8 byte contenente un numero in binario;

NOTA

Queste funzioni sono collegate alle corrispondenti funzioni MKSMBF$ e MKDMBF$.

Vengono utilizzate per file di dati creati con precedenti versioni di BASIC.

ESEMPIO

```
PRINT CVSMBF(BIN4$)
```

DATA

Memorizza le costanti numeriche e di stringa utilizzate dalla istruzione READ.

TIPO Istruzione

SINTASSI

DATA *costante[,costante][,...]*

dove:

costante	è un qualsiasi valore numerico o una costante stringa.

ESEMPIO

```
DATA gennaio, febbraio, marzo, aprile, maggio
DATA giugno, luglio, agosto, settembre, ottobre
DATA novembre, dicembre
```

DATE$

Fornisce una stringa contenente la data corrente.

TIPO Funzione Dos

SINTASSI

DATE$

Fornisce la data con questo formato:
mm - gg - aaaa

ESEMPIO

```
PRINT DATE$
```

DATE$

Imposta la data del giorno corrente.

TIPO Istruzione Dos

SINTASSI

DATE$ = *stringa-data*

dove:

stringa-data espressione stringa in uno dei seguenti formati:

mm-gg-aa
mm-gg-aaaa
mm/gg/aa
mm/gg/aaaa

ESEMPIO

```
DATE$ = Mese$+"/"+Giorno$+"/"+Anno$
```

DECLARE

Dichiara una procedura FUNCTION o SUB e attiva la verifica del tipo di dati dell'argomento.

TIPO Istruzione

SINTASSI

DECLARE FUNCTION|SUB *nome [(elenco-param)]*

dove:

nome nome della procedura

elenco - parametri una o più variabili che specificano i parametri da mandare alla procedura chiamata; si esprimono con la seguente sintassi:

variabile [as tipo][,...];

tipo indica il tipo della variabile e può essere: INTEGER, LONG, SINGLE, DOUBLE o STRING.

ESEMPIO

```
DECLARE SUB Proc-01 (A(2) AS STRING)
```

DIM

Definisce una variabile, un vettore o una matrice assegnando lo spazio in memoria.

TIPO Istruzione

SINTASSI

DIM *[SHARED] variabile[(indici)][AS tipo]...*

dove:

SHARED	Rende la variabile utilizzabile da tutte le procedure del modulo;
variabile	nome della variabile Basic;
indici	espressioni numeriche intere positive che dichiarano le dimensioni della matrice (2 valori separati dalla virgola) o del vettore (un valore);
tipo	dichiara il tipo di variabile (INTEGER, LONG, SINGLE, DOUBLE, STRING o utente).

ESEMPI

```
DIM Num(d%,d%)
DIM I AS INTEGER
```

DEFtipo

Imposta il tipo di dati predefinito delle variabili, delle funzioni DEF FN e delle procedure FUNCTION.

TIPO Istruzione

SINTASSI

DEFINT *l1-l2[,...]*

DEFSNG *l1-l2[,...]*

DEFDBL *l1-l2[,...]*

DEFLNG *l1-l2[,...]*

DEFSTR *l1-l2[,...]*

dove:

l1-l2[,...] indica l'intervallo di lettere da assegnare al tipo di variabile.

I nomi di variabili e di funzioni che iniziano con una lettera compresa nell'intervallo sono del tipo specificato dalle ultime tre lettere del comando:

INT numerico intero

SNG	numerico singola precisione
DBL	numerico doppia precisione
LNG	numerico intero LONG
STR	stringa

ESEMPI

```
DEFDBL A-D
DEFSTR S-Z
```

DO...LOOP

Ripete un blocco di istruzioni fino a che una condizione è vera oppure falsa.

TIPO Istruzione

SINTASSI1

> **DO**
> *[blocco-istruzioni]*
> **LOOP** [WHILE|UNTIL] *condizione*

SINTASSI2

> **DO** [WHILE|UNTIL] *condizione*
> *[blocco istruzioni]*
> **LOOP**

dove:

WHILE	esegue il loop fino a quando la condizione resta vera;
UNTIL	esegue il loop fino a quando la condizione non diventa vera.

ESEMPIO

```
DO
SCELTA$ = INKEY$
LOOP WHILE SCELTA$ = ""
```

DRAW

Disegna un oggetto.

TIPO Istruzione grafica

SINTASSI

DRAW *stringa-comandi*

dove:
stringa-comandi stringa che contiene i comandi per disegnare e colorare l'oggetto (vedi elenco dei comandi con l'help in linea).

ESEMPIO

```
SCREEN 1
DRAW "C2"
DRAW "F60 L120 E60"
DRAW "BD30"
DRAW "P1,2"
```

END

Termina un programma Basic, una procedura o un blocco.

TIPO Istruzione

SINTASSI

END [DEF|FUNCTION|IF|SELECT|SUB|TYPE]

dove:

la mancanza di parametri indica la fine del programma.
L'indicazione di un parametro indica la fine della relativa funzione o struttura.

ESEMPI

```
END

END IF

END SUB
```

ENVIRON

Interfaccia il Dos per modificare un parametro di ambiente.

TIPO Istruzione Dos

SINTASSI

ENVIRON (*espressione-stringa*)

dove:

espressione-stringa ha il formato:

parametro = testo

NOTA

La funzione ENVIRON$ *(stringa-ambiente|n)* estrae dalla tabella di ambiente Dos la stringa indicata dal nome o dal numero progressivo.

ESEMPIO

```
ENVIRON "PATH = A:\ARTICOLI"
```

EOF

Gestisce la condizione di fine file.

TIPO Funzione

SINTASSI

EOF(*n*)

dove:

n	numero di un file aperto

ESEMPIO

```
IF EOF(1) THEN CLOSE:END
```

ERASE

Reinizializza gli elementi di vettori e matrici statiche o disalloca vettori o matrici dinamici.

TIPO Istruzione

SINTASSI

ERASE *nome*

dove:

nome è il nome del vettore o della matrice definito dall'istruzione DIM.

ESEMPIO

```
DIM A(100,100)
.
.
.
ERASE A
```

ERDEV, ERDEV$

Forniscono informazioni sulla condizione di una unità periferica dopo un errore.

TIPO Funzione

SINTASSI

ERDEV fornisce il codice di errore;

ERDEV$ fornisce il nome dell'unità.

ESEMPIO

```
PRINT "ERRORE SULL'UNITA'"; ERDEV$
```

ERL, ERR

Forniscono informazioni sugli errori di programma.

TIPO Funzione

SINTASSI

ERL fornisce il numero di riga;

ERR fornisce il codice di errore.

ESEMPIO

```
PRINT "CODICE DI ERRORE:";ERR
```

ERROR

Simula il verificarsi di un errore Basic o un errore definito dall'utente.

TIPO Istruzione

SINTASSI

ERROR *codice-errore*

dove:

codice-errore	è un numero intero tra 1 e 255.

ESEMPIO

```
ERROR 150
```

EXIT

Controlla l'uscita da un ciclo DO o FOR, da una procedura FUNCTION o SUB o da una funzione DEF FN.

TIPO Istruzione

SINTASSI

EXIT DEF|DO|FOR|FUNCTION|SUB

dove:

DEF uscita da una funzione DEF FN;

DO uscita da un ciclo DO;

FOR uscita da un ciclo FOR;

FUNCTION uscita da una procedura FUNCTION;

SUB uscita da una procedura SUB;

ESEMPIO

```
DO
EXIT DO
```

EXP

Fornisce il valore della funzione esponenziale:
e elevato alla potenza indicata nell'argomento.

TIPO Funzione matematica

SINTASSI

EXP(*espressione-numerica*)

dove:

espressione-numerica esprime la potenza; deve avere un valore non superiore a 88,02969.

ESEMPIO

```
PRINT EXP(1)
```

FIELD

Assegna spazio alle variabili nel buffer di un file ad accesso RANDOM.

TIPO Istruzione

SINTASSI

FIELD *n-file, n-caratteri* AS *var-stringa [,...]*

dove:

n-file numero di un file aperto;

n-caratteri numero di caratteri contenuti;

n-stringa nome della variabile stringa.

ESEMPIO

```
OPEN "Articoli" FOR RANDOM AS #1 LEN 58
FIELD #1,10 AS Cod$, 40 AS Des$, 8 AS Pu$
```

FILEATTR

Fornisce informazioni su un file aperto

TIPO Funzione

SINTASSI

FILEATTR(*n-file, attributo*)

dove:

n-file	numero di un file aperto;
attributo	indica il tipo di informazione da fornire:

1 modalità di accesso al file (input, output, ...)
2 nome Dos del file.

ESEMPIO

```
PRINT FILEATTR(1,2)
```

FILES

Visualizza i nomi dei file sulla directory corrente o specificata (comando Dos).

TIPO Istruzione Dos

SINTASSI

FILES *spec-file*

dove:

spec-file	stringa che indica un nome di file (esprimibile anche con i caratteri jolly) ed eventualmente un PATH; se omesso vengono visualizzati tutti i file della directory corrente.

ESEMPI

```
FILES

FILES "A:\BASIC\*.BAS"
```

FIX

Fornisce la parte intera troncata di una espressione numerica.

TIPO Funzione matematica

SINTASSI

FIX (*espressione-numerica*)

dove:

espressione-numerica qualsiasi espressione numerica.

ESEMPIO

```
PRINT FIX(85.48)
PRINT FIX(Numero)
```

FOR...NEXT

Ripete un gruppo di istruzioni un determinato numero di volte.

TIPO Istruzione

SINTASSI

FOR *contatore = n-inizio TO n-fine [*STEP *incr]*
NEXT *[contatore] [,...]*

dove:

contatore variabile numerica intera usata come contatore del ciclo;

n-inizio e *n-fine* valori iniziale e finale del contatore;

incr incremento del contatore dopo ogni iterazione; l'incremento di default è +1.

ESEMPIO

```
FOR N% = 1 TO 23
LOCATE N%,80: PRINT "*"
NEXT N%
```

FRE

Fornisce la quantita' di memoria disponibile.

TIPO Funzione

SINTASSI 1

FRE(*espressione-numerica*)

SINTASSI 2

FRE(*espressione-stringa*)

dove:

espressione-numerica valore che specifica il tipo di memoria:

-1 fornisce le dimensioni della matrice non stringa piu' grande che e' possibile creare;

-2 fornisce lo spazio di stack disponibile al programma;

espressione-stringa costante o variabile stringa qualsiasi: fornisce la quantita' di spazio stringa disponibile.

ESEMPI

```
PRINT "Spazio stringa:"; FRE("")
PRINT "Spazio della matrice:"; FRE(-1)
```

FREEFILE

Fornisce il successivo numero di file Basic libero.

TIPO Istruzione

SINTASSI

FREEFILE

ESEMPIO

```
PRINT "Numero successivo di file:"; FREEFILE
```

FUNCTION

Definisce una procedura FUNCTION.

TIPO Istruzione

SINTASSI

FUNCTION *nome[(elenco-parametri)]* [STATIC]
[blocco istruzioni]
END FUNCTION

dove:

nome nome della funzione;

elenco-parametri una o più variabili che indicano i parametri da passare alla funzione quando viene chiamata; si esprimono con la seguente sintassi:

*variabile [*AS *tipo][,...]*

tipo indica il tipo della variabile e può essere: INTEGER, LONG, SINGLE, DOUBLE, o STRING;

STATIC specifica che i valori delle variabili locali della funzione vengono salvati tra una chiamata e l'altra.

ESEMPIO

```
FUNCTION Calcola Prezzo AS DOUBLE
      .
      .
      .
END FUNCTION
```

GET

Legge i dati da un file ad accesso RANDOM.

TIPO Istruzione I/O

SINTASSI

GET *n-file[, n-rec][, variabile]*

dove:

n-file numero di un file aperto;

n-rec numero del record da leggere; deve essere un intero positivo:
max 2^31 = 2.147.483.647;
se omesso viene letto il record successivo;

variabile variabile utilizzata per ricevere l'input dal file; se omessa occorre utilizzare l'istruzione FIELD e i convertitori numerici (CVS, CVD, CVL, CVI).

ESEMPIO

```
GET #1, I%, RECORD
```

GET(grafica)

Memorizza un'immagine grafica prelevata dallo schermo.

TIPO Istruzione grafica

SINTASSI

GET [STEP] *(x1,y1)* - [STEP] *(x2,y2), matr [ind]*

dove:

STEP consente l'utilizzo di coordinate di schermo relative;

x1,y1,x2,y2 coordinate degli angoli diametralmente opposti del riquadro da memorizzare;

matr nome della matrice in cui memorizzare l'immagine;

ind indici di matrice: se indicati censentono di memorizzare l'immagine da un elemento di matrice diverso dal primo.

ESEMPIO

```
GET (140,25) - (240,125), cubo
```

GOSUB..RETURN

Passa il controllo a una subroutine e gestisce il ritorno.

TIPO Istruzione

SINTASSI

GOSUB *etichetta1|riga1*
.
.
RETURN *[etichetta2|riga2]*

dove:

etichetta1,riga1 etichetta o numero di riga di inizio della subroutine;

etichetta2,riga2 etichetta o numero di riga a cui ritornare; se omesso ritorna all'istruzione successiva alla GOSUB.

ESEMPIO

```
GOSUB Calcola

Calcola:

RETURN
```

GOTO

Effettua un salto incondizionato alla riga specificata.

TIPO Istruzione

SINTASSI

GOTO *etichetta|riga*

dove:

etichetta,riga etichetta o numero di riga a cui passare il controllo.

ESEMPIO

```
GO TO INIZIO

GO TO 10
```

HEX$

Fornisce una stringa contenente la rappresentazione esadecimale dell'argomento decimale.

TIPO Funzione stringa

SINTASSI

HEX$(*espressione-numerica*)

dove:

espressione - numerica espressione numerica con un valore decimale.

ESEMPIO

```
A$ = HEZ$(X)
PRINT X; "Decimale = "; A$; "Esadecimale"
```

IF...THEN...ELSE

Esegue un' istruzione o un blocco di istruzioni a seconda del valore vero o falso di una condizione.

TIPO Istruzione

SINTASSI1

IF *condizione then azione-v [*ELSE *azione-f]*

dove:

condizione espressione di confronto con valore VERO (1) o FALSO (0);

azione-v una o più istruzioni monoriga da eseguire se la condizione è vera;

azione-f una o più istruzioni monoriga da eseguire se la condizione è falsa; se viene omesso ELSE il controllo per condizione falsa passa alla riga successiva.

SINTASSI2

IF *condizione1* THEN
[blocco-istruzioni-1]
*[*ELSEIF *condizione2* THEN
[blocco-istruzioni-2]]

```
[ELSE
[blocco-istruzioni-n]]
END IF
```

dove:

condizione1,condizione2 espressioni di confronto;

blocco-istruzioni una o più istruzioni BASIC monoriga o multiriga.

NOTA

Per maggiori approfondimenti sulla struttura di questa istruzione si consulti la guida in linea attivando la voce *DETTAGLI*.

ESEMPIO

```
IF X < 10 THEN
    Y=1
ELSEIF X < 100 THEN
    Y=2
ELSEIF X < 1000 THEN
    Y=3
ELSELF X < 10000 THEN
    Y=9
ELSE
    Y=5
END IF
PRINT "Il numero analizzato ha ";Y:" cifre"
```

INKEY$

Legge un carattere dalla tastiera.

TIPO Funzione I/O

SINTASSI

INKEY$

ESEMPIO

```
PRINT "Premere ESC per uscire"
DO
LOOP UNTIL INKEY$ = CHR$(27)
REM       27 = codice Ascii di ESC
```

INP

Fornisce un byte letto da una porta di I/O.

TIPO Funzione di I/O

SINTASSI

INP(*porta*)

dove:

porta	numero intero identificativo della porta (max 65.535).

ESEMPIO

```
X% = INP(&H3FC)
OUT &H3FC,(X%XOR1)
```

INPUT

Legge i dati dalla tastiera.

TIPO Istruzione I/O

SINTASSI

INPUT*[;] ["stringa";/,] variabile[,...]*

dove:

stringa	viene visualizzata sullo schermo prima del PROMPT per richiedere i dati;
variabile	variabile di memoria in cui vengono memorizzati i dati di input;
;	il punto e virgola subito dopo INPUT non porta a capo il cursore dopo aver premuto INVIO; posto dopo la stringa visualizza la stessa con un punto di domanda;
,	la virgola dopo la stringa non visualizza il punto di domanda.

ESEMPIO

```
INPUT "IMMETTI NOMINATIVO:  ",NOME$
```

INPUT

Legge i dati da un file sequenziale.

TIPO Istruzione I/O

SINTASSI

INPUT #*n-file, elenco-variabili*

dove:

n-file numero di un file aperto in INPUT;

elenco-variabili una o più variabili separate da virgola in funzione della struttura del record.

ESEMPIO

```
INPUT #1, NOME$, INDIRIZZO$, TELEFONO$
```

INPUT$

Fornisce una stringa di caratteri letta dal file specificato.

TIPO Funzione I/O

SINTASSI

INPUT$*(n[, n-file])*

dove:

n	numero di caratteri da leggere;
n-file	numero di un file aperto in INPUT; se viene omesso legge i dati da tastiera.

ESEMPIO

```
PRINT INPUT$(10,1)
```

INSTR

Fornisce la posizione della prima occorrenza di una stringa all'interno di un'altra stringa.

TIPO Funzione

SINTASSI

INSTR(*[inizio,] stringa1, stringa2)*

dove:

inizio	posizione del carattere da cui iniziare la ricerca; se omesso inizia da 1;
stringa1	stringa in cui effettuare la ricerca;
stringa2	stringa da cercare.

ESEMPIO

```
PRINT    "posizione della parola BASIC = ";
                        INSTR(F$,"BASIC")
```

INT

Fornisce la sola parte intera di un numero decimale.

TIPO Funzione matematica

SINTASSI

INT(*espressione-numerica*)

dove:

espressione-numerica qualsiasi espressione numerica.

ESEMPIO

```
PRINT INT(N)
```

IOCTL

Trasmette una stringa di controllo ad un Driver di unità periferica.

TIPO Istruzione I/O

SINTASSI

IOCTL *n-file, stringa*

dove:

n-file	numero di una periferica aperta;
stringa	stringa contenente il comando da inviare.

ESEMPIO

```
IOCTL #1, "RAW"
```

IOCTL$

Legge una stringa di controllo da un Driver di unità periferica.

TIPO Funzione I/O

SINTASSI

IOCTL$*(n-file)*

dove:

n-file numero di una periferica aperta.

ESEMPIO

```
IF IOCTL$(1) = "0" THEN CLOSE 1
```

KEY

Assegna valori di stringa ai tasti funzionali, li visualizza, attiva o disattiva la riga di visualizzazione dei tasti funzionali.

TIPO Istruzione

SINTASSI

KEY *n, stringa*

KEY LIST|ON|OFF

dove:

n	numero di un tasto funzione;
stringa	non più di 15 caratteri abbinati al tasto funzione;
LIST	visualizza gli abbinamenti;
ON|OFF	attiva/disattiva la riga di visualizzazione dei tasti programmati in fondo allo schermo.

ESEMPI

```
KEY 1, "ESCI"
KEY ON
```

KEY(n)

Attiva, disattiva o sospende la gestione degli eventi di un tasto funzionale.

TIPO Istruzione

SINTASSI

KEY*(n)* ON|OFF|STOP

dove:

n numero di un tasto funzione;

ON|OFF|STOP attivazione, disattivazione, sospensione dell'evento.

ESEMPIO

KEY(7) ON

KILL

Elimina un file da un disco.

TIPO Istruzione

SINTASSI

KILL *spec-file*

dove:

spec-file	stringa con il nome del file ed eventualmente PATH e caratteri jolly.

ESEMPIO

```
KILL "A:\Basic\Rubrica.Seq"
```

LBOUND

Fornisce il limite inferiore della dimensione indicata di una matrice.

TIPO Funzione

SINTASSI

LBOUND(*matrice[,dim]*)

dove:

matrice	nome della matrice;
dim	numero intero che indica di quale dimensione della matrice fornire il limite inferiore;parametro opzionale per vettori (matrici unidimensionali).

ESEMPIO

```
DIM A(1 TO 100, 0 TO 50, -3 TO 4)
LBOUND(A, 1)          'FORNISCE  1
LBOUND(A, 2)          'FORNISCE  0
LBOUND(A, 3)          'FORNISCE  -3
```

LCASE$

Converte in minuscolo tutti i caratteri di una stringa.

TIPO Funzione stringa

SINTASSI

LCASE$*(stringa)*

dove:

stringa qualsiasi espressione a stringa.

ESEMPIO

```
PRINT LCASE$(descrizione$)
```

LEFT$

Fornisce il numero indicato di caratteri di una stringa a partire da sinistra.

TIPO Funzione stringa

SINTASSI

LEFT$*(stringa,l)*

dove:

stringa	qualsiasi espressione a stringa;
l	numero di caratteri da fornire.

ESEMPIO

```
PRINT LEFT$("Dottore",4)        'scrive Dott
```

LEN

Fornisce il numero di caratteri di una stringa o il numero di byte di una variabile non stringa.

TIPO Funzione

SINTASSI

LEN(*stringa*|*variabile*)

dove:

stringa	qualsiasi espressione a stringa;
variabile	variabile non a stringa.

ESEMPIO

```
LOCATE 3,(40 - LEN(tit$)/2)      'scrive il titolo
PRINT tit$                       'centrato
```

LET

Assegna il valore di una espressione a una variabile.

TIPO Istruzione

SINTASSI

[LET] *variabile* = *espressione*

dove:

variabile qualsiasi variabile;

espressione qualsiasi espressione.

NOTA

Se si omette LET l'istruzione di assegnazione ha lo stesso effetto.

ESEMPI

LET N = 10 oppure N = 10

LET A$ = "ROSSI" oppure A$ = "ROSSI"

LINE

Disegna una linea o un riquadro sullo schermo.

TIPO Istruzione grafica

SINTASSI

LINE *[[*STEP*] (x1, y1)] - [*STEP*](x2, y2)[, [colore] [, [*B*[*F*][,stile]]]*

dove:

STEP	indica che le coordinate sono relative alla posizione corrente del cursore;
x1 ,y1	coordinate di inizio del segmento;
x2, y2	coordinate di fine del segmento;
colore	attributo di colore della riga o del riquadro;
B	traccia un riquadro;
BF	traccia un riquadro colorato;
stile	valore a 16 Bit che stabilisce se alcuni pixel debbano essere tracciati o no; si utilizza per

tracciare righe intermittenti o punteggiate.

ESEMPIO

```
SCREEN 1
LINE (110,70) - (190,120),,B
```

LINE INPUT

Legge una riga da tastiera (max 255 caratteri) senza utilizzare delimitatori.

TIPO Istruzione I/O

SINTASSI

LINE INPUT*[;] ["stringa";|,] var-stringa*

dove:

stringa viene visualizzata sullo schermo prima del PROMPT per richiedere i dati;

var-stringa variabile a stringa in cui vengono memorizzati i dati di input;

; il punto e virgola subito dopo LINE INPUT non porta a capo il cursore dopo aver premuto INVIO;

, la virgola dopo la stringa non visualizza il punto di domanda.

NOTA

Si utilizza LINE INPUT quando si debbono memorizzare caratteri (come la virgola) che l'istruzione INPUT interpreta invece come delimitatori.

ESEMPIO

```
LINE INPUT "Immetti testo: ",Riga$
```

LINE INPUT

Legge una stringa di dati senza delimitatori da un file sequenziale.

TIPO Istruzione I/O

SINTASSI

LINE INPUT *#n-file, var-stringa*

dove:

n-file numero di un file aperto in INPUT;

var-stringa variabile stringa in cui vengono memorizzatii dati del record. La fine di ogni record è indicata dalla sequenza di caratteri A CAPO - INTERLINEA.

ESEMPIO

```
LINE INPUT #1, RECORD$
```

LOC

Fornisce la posizione corrente all'interno di un file.

TIPO Funzione I/O

SINTASSI

LOC(*n-file)*

dove:

n-file numero di un file aperto.

ESEMPIO

```
GET #1,,record
PRINT "record numero";LOC(1)
```

LOCATE

Posiziona il cursore nel punto specificato.

TIPO Istruzione I/O

SINTASSI

LOCATE *[r][,c][, [flag-curs][, [inizio, stop]]]*

dove:

r, c	numeri di riga e colonna; se omessi vengono assunti i valori correnti;
flag - curs	se è 0 (zero) indica che il cursore è acceso (visibile);
inizio, stop	valori interi compresi tra 0 e 31 che indicano le righe di pixel che formano il cursore, determinandone la dimensione.

ESEMPIO

```
LOCATE 10,15,,1,31
```

LOCK...UNLOCK

Impedisce e ripristina l'accesso a parte o a tutto un file aperto.

TIPO Istruzione I/O

SINTASSI

LOCK|UNLOCK *n-file[, record|* *[inizio]* TO *fine]*

dove:

n-file	numero di un file aperto;
record	numero del record da bloccare/sbloccare (per file ad acceso RANDOM);
inizio, fine	numeri del primo e dell'ultimo record da bloccare/sbloccare.

NOTA
Per i file sequenziali LOCK e UNLOCK agiscono su tutto il file.

ESEMPIO

```
LOCK #1,2          'Blocca il record 2
GET #1,2,REC
UNLOCK #1,2            'Sblocca il record 2
```

LOF

Fornisce la lunghezza in byte del file specificato.

TIPO Funzione I/O

SINTASSI

LOF(*n-file*)

dove:

n-file numero di file aperto.

ESEMPIO

```
PRINT "Il file e' lungo ";LOF(1);"byte"
```

LOG

Fornisce il logaritmo naturale di una espressione numerica.

TIPO Funzione matematica

SINTASSI

LOG*(n)*

dove:

n	espressione numerica che esprime il numero di cui è richiesto il logaritmo naturale.

ESEMPIO

```
A = LOG(B)
```

LPOS

Fornisce la posizione corrente della testina di scrittura della stampante all'interno del buffer della stampante.

TIPO Funzione I/O

SINTASSI

LPOS(*n*)

dove:

n	espressione numerica con valore da 0 a 3 che indica la porta della stampante (LPT1=O/1, LPT2=2,...).

ESEMPIO

```
IF LPOS(0)>50 THEN
    LPRINT
ELSE
    LPRINT ",";
END IF
```

LPRINT

Scrive i dati sulla stampante LPT1.

TIPO Istruzione I/O

SINTASSI

LPRINT *[*USING *stringa-formato;]*
elenco-espressioni[;|,]

dove:

elenco - espressioni una o più variabili o costanti contenenti i dati da scrivere;

;|, sia la virgola che il punto e virgola indicano di non effettuare il comando di A CAPO in modo che il successivo output avvenga sulla stessa riga;

USING permette la formattazione dell' OUTPUT;

stringa-formato espressione a stringa contenente i caratteri indicatori di formato.

ESEMPI

```
LPRINT "INDIRIZZO:",IND$

LPRINT USING "###.##"; NUM
```

LSET

Trasferisce i dati dalla memoria al buffer di un file RANDOM (prima dell'istruzione PUT), allinea a sinistra il valore di una variabile stringa, o copia una variabile di record in un'altra.

TIPO Istruzione

SINTASSI1

LSET *var-stringa = espressione-stringa*

SINTASSI2

LSET *var-rec1 = var-rec2*

dove:

var-stringa variabile stringa o campo di un file RANDOM definito nell'istruzione FIELD;

espressione-stringa qualunque espressione stringa;
var-rec1, var-rec2 variabili di record.

ESEMPI

```
LSET A$ = MKD$(A#)
LSET REC_A = REC_B
```

LTRIM$

Elimina gli spazi iniziali da una stringa.

TIPO Funzione stringa

SINTASSI

LTRIM$*(espressione-stringa)*

dove:

espressione-stringa qualunque espressione stringa.

ESEMPIO

```
A$ = "    ROSSI Giovanni"
PRINT LTRIM$(A$)
```

MID$

Fornisce il numero indicato di caratteri di una stringa a partire da una certa posizione.

TIPO Funzione stringa

SINTASSI

MID$(*stringa, n[,l]*)

dove:

stringa	qualsiasi espressione a stringa;
n	posizione del primo carattere da estrarre;
l	numero do caratteri da fornire.

ESEMPIO

```
PRINT MID$ ("ITALIANO",3,3)    'scrive ALI
```

MID$

Sostituisce una parte intermedia di una variabile stringa con un' altra stringa.

TIPO Istruzione

SINTASSI

MID$*(var-stringa, inizio[,lunghezza]) = stringa*

dove:

var - stringa variabile stringa da modificare;

stringa qualunque espressione a stringa che fornisce i nuovi caratteri;

inizio espressione numerica intera che indica la posizione del primo carattere da sostituire;

lunghezza numero di caratteri da sostituire (da sinistra); se omesso il parametro *stringa* viene utilizzato per intero.

ESEMPIO

```
A$ = "ANNIBALE ROSSI"
MID$ (A$,2) = "MILCAR"
PRINT A$            'scrive AMILCARE
```

MKI$, MKL$, MKS$, MKD$

Convertono in valori di stringa valori numerici. Vengono utilizzate con le istruzioni FIELD e PUT per scrivere valori numerici su un file RANDOM.

TIPO Funzione

SINTASSI

MKI$(*n-i*) *converte in stringa a 2 byte;*

MKL$(*n-l*) *converte in stringa a 4 byte;*

MKS$(*n-s*) *converte in stringa a 4 byte;*

MKD$(*n-d*) *converte in stringa a 8 byte;*

dove:

n-i espressione numerica intera;

n-l espressione numerica long;

n-s espressione numerica a semplice precisione;

n-d espressione numerica a doppia precisione.

NOTA

Queste funzioni sono collegate alle corrispondenti funzioni CVI, CVL, CVS, CVD, per convertire una stringa in numero .

ESEMPIO

```
LSET N$ = MKD$(N#)
```

MKDIR

Crea una nuova directory.

TIPO Istruzione Dos

SINTASSI

MKDIR *nome-dir*

dove:

nome-dir espressione a stringa che identifica la directory.

ESEMPIO

```
MKDIR "PROVA"
```

MKDMBF$, MKSMBF$

Convertono numeri in formato IEEE in stringhe numeriche in formato binario Microsoft.
Vemgono utilizzate con le istruzioni FIELD e PUT per i file RANDOM.

TIPO Funzione

SINTASSI

MKDMBF$*(n-d)* converte in stringa di 8 byte;

MKSMBF$*(n-s)* converte in stringa di 4 byte;

dove:

n-d, n-s espressioni numeriche a doppia o a semplice precisione in formato IEEE.

NOTA
Queste funzioni sono collegate alle corrispondenti funzioni CVDMBF e CVSMBF.
Vengono utilizzate per file di dati creati con precedenti versioni di BASIC.

ESEMPIO

```
NB$ = MKSMBF$ (NS)
```

NAME

Rinomina un file o una directory.

TIPO Istruzione

SINTASSI

NAME *file-prec* AS *file-nuovo*

dove:

file-prec nome del file esistente;

file-nuovo nome nuovo da assegnare al file.

ESEMPIO

```
NAME "FILE-PROVV" AS "FILE-ART"

NAME FILEOLD$ AS FILENEW$
```

OCT$

Fornisce una stringa con il valore ottale dell'argomento numerico.

TIPO Funzione

SINTASSI

OCT$(*n*)

dove:

n	espressione numerica da convertire in ottale; se decimale viene arrotondata.

ESEMPIO

```
PRINT "Il valore ottale di";N%;"e'";OCT$(N%)
```

ON evento

Al verificarsi dell'evento indicato passa il controllo ad una subroutine.

TIPO Istruzione

SINTASSI

ON *evento* GOSUB *sub*

dove:

sub etichetta o numero di riga di inizio della subroutine;

evento può essere indicato uno dei seguenti eventi:

COM(*n*) caratteri in arrivo alla porta *n*;

KEY(*n*) pressione del tasto *n*;

PEN azione di penna ottica;

PLAY(*n*) meno di *n* note nel buffer musica di sottofondo;

STRING(*n*) azione del pulsante *n* del joystick;

TIMER(*n*) tempo trascorso.

ESEMPIO

```
TIMER ON                  'Attiva il timer
ON TIMER(5) GOSUB VIA     'Passa il controllo
                          'dopo 5 secondi
```

ON ERROR

Attiva la gestione degli errori di timeout.

TIPO Istruzione

SINTASSI

ON ERROR GOTO *rout* | RESUME NEXT

dove:

rout etichetta o numero di riga di inizio della routine di gestione errori;

RESUME NEXT riprende l'esecuzione dall'istruzione successiva a quella in cui si è verificato l'errore di timeout.

ESEMPIO

```
ON ERROR GOTO GESTERR
```

ON...GOSUB/GOTO

A seconda del valore di un'espressione numerica passa il controllo ad una delle righe specificate.

TIPO Istruzione

SINTASSI

ON *n* GOSUB|GOTO *elenco-etichette*

dove:

n qualsiasi espressione numerica; se il valore e' decimale viene arrotondato;

elenco-etichette lista di etichette o numeri riga separati da virgole; il controllo viene passato all'etichetta/numero-riga, la cui posizione corrisponde al valore di espressione.

NOTA
Se il valore di espressione è 0 o maggiore del numero di parametri il controllo passa alla successiva istruzione Basic.

ESEMPIO

```
ON scelta% GOSUB A1,A2,A3,A4,A5
```

OPEN

Apre un file.

TIPO Istruzione I/O

SINTASSI

OPEN *file* FOR *mod [ACCESS tipo] [blocco]*
AS n-file [LEN = l-rec]

dove:

file	espressione stringa che indica il nome del file ed eventualmente PATH e DRIVE;
mod	modalità di apertura (APPEND, BYNARY, INPUT, OUTPUT o RANDOM);
tipo	specifica, in ambiente di rete, se il file viene aperto per un accesso di tipo READ (lettura), WRITE (scrittura) o READ WRITE (entrambi);
blocco	specifica, in ambiente di rete, il tipo di blocco del file (SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE);

n-file	numero identificativo del file;
l-rec	per i file RANDOM indica la lunghezza del record (se omesso assume 128 byte), per i file sequenziali le dimensioni del buffer di I/O (se omesso assume 512 byte).

ESEMPIO

```
OPEN "ARTICOLI.RAN" FOR RANDOM AS #1
```

OPTION BASE

Imposta il limite inferiore predefinito per gli indici di una matrice.

TIPO Istruzione

SINTASSI

OPTION BASE *n*

dove:

n espressione numerica - intera.

NOTA

Si può impostare anche con l'istruzione DIM (si consulti la guida in linea).

ESEMPIO

```
OPTION BASE 1
```

OUT

Invia un byte ad una porta di I/O.

TIPO Istruzione I/O

SINTASSI

OUT *porta, dato*

dove:

porta	espressione numerica intera che identifica la porta;
dato	espressione numerica intera da inviare.

ESEMPIO

```
OUT &H3FC,X%
```

PAINT

Riempie un'area grafica con il colore o il motivo specificato.

TIPO Istruzione grafica

SINTASSI

PAINT *[STEP] (x,y)[, [colore][, [bordo][,sfondo]]]*

dove:

x,y	coordinate di un punto dello schermo che indica la zona da colorare;
STEP	indica che le coordinate sono relative alla posizione corrente del cursore;
colore	espressione numerica che indica un attributo di colore o stringa che indica il motivo;
bordo	espressione numerica che indica un attributo di colore per il bordo della figura;

sfondo	stringa che specifica la corrispondenza tra lo sfondo della figura e il motivo della casella.

NOTA

Per la tabella dei colori si veda l'istruzione COLOR.

ESEMPIO

```
PAINT (50,100), 1, 2    'colora in azzurro
```

PALETTE

Modifica uno o più colori della PALETTE.

TIPO Istruzione grafica

SINTASSI1 (per l'assegnazione di un colore)

PALETTE *[attributo, colore]*

SINTASSI2 (per l'assegnazione di più colori)

PALETTE USING *nome-matrice[(indice)]*

dove:

attributo indica la PALETTE da modificare (0 - 1);

colore numero del colore da assegnare;

nome-matrice matrice contenente i numeri dei colori da assegnare;

indice indice del primo elemento della matrice da assegnare.

ESEMPI

```
PALETTE 0,2
PALETTE USING A%(0)
```

PCOPY

Copia una pagina di schermo in un'altra.

TIPO Istruzione grafica

SINTASSI

PCOPY *pag1, pag2*

dove:

pag1	espressione numerica intera che indica la pagina di memoria video da copiare;
pag2	indica la pagina in cui copiare.

ESEMPIO

```
PCOPY 3,5
```

PEEK

Fornisce il valore del byte memorizzato nella posizione di memoria specificata.

TIPO Funzione

SINTASSI

PEEK(*indirizzo*)

dove:

indirizzo	numero di byte relativo all'indirizzo impostato da DEF SEG.

ESEMPIO

```
DEF SEG = 0
STATO% = PEEK(&H417)
```

PEN

Fornisce le coordinate della penna ottica.

TIPO Funzione

SINTASSI

PEN(*n*)

dove:

n	espressione numerica intera tra 0 e 9 che indica il tipo di informazione da fornire secondo la seguente tabella:

0 penna appoggiata o no (-1 = si; 0 = no);

1 coordinata x dell'ultima pressione;

2 coordinata y dell'ultima pressione;

3 stato corrente (-1 = sù; 0 = giù);

4 coordinata x dell'ultima posizione nota;

5 coordinata y dell'ultima posizione nota;

6 riga dell'ultima pressione;

7 colonna dell'ultima pressione;

8 riga dell'ultima pressione nota;

9 colonna dell'ultima posizione nota.

ESEMPIO

```
x = PEN(4):y = PEN(5)
```

PEN

Attiva, disattiva o sospende la gestione degli eventi della penna ottica.

TIPO Istruzione

SINTASSI

PEN ON|OFF|STOP

dove:

ON	attiva il gestore degli eventi della penna ottica definito in un'istruzione ON PEN;
OFF	disattiva;
STOP	sospende.

ESEMPIO

```
PEN ON
```

PLAY

Fornisce il numero di note non ancora suonate della coda di musica di sottofondo.

TIPO Funzione musicale

SINTASSI

PLAY*(n)*

dove:

n qualsiasi espressione numerica.

ESEMPIO

```
PLAY MOTIVO$
IF PLAY(1) = 10 THEN
    PRINT "Ancora pochi secondi"
END IF
```

PLAY

Suona delle note musicali.

TIPO Istruzione musicale

SINTASSI

PLAY *note*

dove:

note stringa contenente la sequenza dei comandi musicali da eseguire, come ottava, note, pause, tempi, modalità; si consulti la guida in linea per conoscere i singoli comandi.

ESEMPIO

```
SCALA$ = "CDEFGAB"
PLAY SCALA$
```

PLAY evento

Attiva, disattiva o sospende la gestione degli eventi di suono.

TIPO Istruzione musicale

SINTASSI

PLAY ON|OFF|STOP
ON PLAY*(limite) GOSUB sub-play*

dove:

OFF|ON|STOP attiva/disattiva/sospende la gestione degli eventi di suono;

limite valore numerico intero da 1 a 32; il controllo viene passato alla subroutine indicata quando nel buffer è contenuto un numero di note inferiore a *limite*;

sub - play subroutine di gestione degli eventi.

ESEMPIO

```
PLAY ON
ON PLAY(5) GOSUB Sotto
```

POINT

Fornisce le coordinate correnti del cursore grafico o il colore di un pixel specificato.

TIPO Funzione grafica

SINTASSI

POINT*(n)* **|** *(x,y)*

dove:

n	valore numerico intero da 0 a 3 che indica la modalità con cui viene fornita la posizione del cursore grafico (vedi guida in linea);
x,y	coordinate del pixel di cui si richiede il colore.

ESEMPIO

```
PRINT POINT (x,y)
```

POKE

Scrive un byte nella posizione di memoria specificata.

TIPO Istruzione

SINTASSI

POKE *indirizzo, byte*

dove:

indirizzo	numero di byte relativo all'indirizzo impostato da DEF SEG.
byte	valore numerico intero tra o e 255 che indica (in codice ASCII) il byte da scrivere.

ESEMPIO

POKE &H417

POS

Fornisce la colonna su cui si trova il cursore.

TIPO Funzione

SINTASSI

POS*(n)*

dove:

n espressione numerica qualsiasi.

ESEMPIO

```
PRINT "Il cursore e' aqlla colonna "; POS(0)
```

PRESET (PSET)

Disegna sullo schermo il punto specificato.

TIPO Istruzione

SINTASSI

PRESET *[*STEP*] (x, y)[, colore]*

dove:

STEP	indica che le coordinate sono relative alla posizione corrente del cursore;
x, y	coordinate del punto dello schermo in cui si vuole disegnare il pixel colorato.
colore	attributo di colore.

ESEMPIO

```
PRESET (X%,100), 5
```

PRINT

Scrive i dati sullo schermo.

TIPO Istruzione I/O

SINTASSI

PRINT *[*USING *stringa-formato;]*
elenco-espressioni[;|,]

dove:

elenco-espressioni una o più variabili o costanti contenenti i dati da scrivere;

;|, indica di non effettuare il comando di A CAPO; il successivo output avverra' sulla stessa riga;

USING permette la formattazione dell'output;

stringa-formato espressione a stringa contenente i caratteri indicatori di formato.

ESEMPIO

```
PRINT "NOMINATIVO:";NOME$
PRINT USING "###.###";QTA
```

PRINT

Scrive i dati su un file sequenziale.

TIPO Istruzione I/O

SINTASSI

PRINT #*n-file, elenco-espressioni*

dove:

n-file numero di un file aperto in output;

elenco-espressioni una o più espressioni separate da virgola che formano la struttura del record da scrivere.

ESEMPIO

```
PRINT #2, NOME$, INDIRIZZO$, TELEFONO$
```

PUT

Scrive i dati su un file ad accesso RANDOM.

TIPO Istruzione I/O

SINTASSI

PUT *n-file[, n-rec][, variabile]*

dove:

n-file	numero di un file aperto;
n-rec	numero del record da scrivere; deve essere un intero positivo (max 2^31 - 1 = 2.147.483.647);
variabile	variabile utilizzata per memorizzare i dati del record da scrivere; se omessa occorre utilizzare l'istruzione FIELD e i convertitori numerici (MKI$, MKL$, MKS$, MKD$).

ESEMPIO

```
PUT #1, I%, Record
```

PUT (grafica)

Visualizza un'immagine grafica memorizzata con l'istruzione GET.

TIPO Istruzione grafica

SINTASSI

PUT *[STEP] (x1, y1), matrice[(indici)][, azione]*

dove:

STEP	consente l'utilizzo di coordinate di schermo relative;
x1, y1	coordinate dell'angolo in alto a sinistra ad indicare la posizione dove collocare l'immagine;
matrice	nome della matrice in cui è memorizzata l'immagine;
indici	se indicati consentono di prelevare l'immagine da un elemento di matrice diverso dal primo;
azione	indica in che modo deve essere visualizzata l'immagine; le opzioni possibili sono:

AND Unisce l'immagine memorizzata con una precedente;

OR Sovrappone l'immagine memorizzata all'immagine esistente;

PSET Disegna l'immagine memorizzata cancellando la precedente;

PRESET Disegna l'immagine memorizzata invertendo i colori e cancellando la precedente;

XOR Disegna l'immagine memorizzata o cancella la precedente conservando lo sfondo; viene utilizzata per produrre effetti di animazione.

ESEMPIO

```
PUT(140,25),CUBO,XOR
```

RANDOMIZE

Inizializza il generatore di numeri casuali.

TIPO Istruzione matematica

SINTASSI

RANDOMIZE *[seme]*

dove:

seme	espressione numerica intera di inizializzazione; se omesso viene richiesto da tastiera.

ESEMPI

```
RANDOMIZE n%

RANDOMIZE TIMER
```

READ

Legge i dati di un'istruzione DATA assegnandoli alle variabili.

TIPO Istruzione I/O

SINTASSI

READ *variabile[,...]*

dove:

variabile di stringa o numerica.

ESEMPIO

```
DATA "Rossi Luigi","via Roma 5","011/554433"
READ NOME$,INDIRIZZO$,TELEFONO$
```

REDIM

Modifica lo spazio assegnato ad una matrice dichiarata $DYNAMIC.

TIPO Istruzione

SINTASSI

REDIM *[*SHARED*] variabile(indici) [*AS *tipo][,...]*

dove:

SHARED rende la variabile utilizzabile da tutte le procedure del modulo;

variabile nome della variabile Basic;

indici espressioni numeriche intere positive che dichiarano le dimensioni della matrice (2 valori separati dalla virgola);

tipo dichiara il tipo di variabile (INTEGER, LONG, SINGLE, DOUBLE, STRING o di tipo utente).

ESEMPIO

```
DIM A$(50,50)
REDIM A$(20,20)
```

REM

Permette l'inserimento di commenti in un programma.

TIPO Istruzione

SINTASSI

REM *commento*

dove:

commento un testo qualsiasi.

NOTA

Per inserire un commento si può anche utilizzare il simbolo apice (') che deve precedere il testo.

ESEMPIO

```
REM     QUESTO E' UN COMMENTO
'       ANCHE QUESTO
```

RESET

Chiude tutti i file aperti.

TIPO Istruzione I/O

SINTASSI

RESET

ESEMPIO

```
RESET
```

RESTORE

Permette ad una istruzione READ di rileggere i dati di una istruzione DATA.

TIPO Istruzione

SINTASSI

RESTORE *[etichetta | riga]*

dove:

etichetta | riga etichetta o numero di riga dell'istruzione DATA; se omessa viene assunta la prima istruzione DATA incontrata nel programma.

ESEMPIO

RESTORE Leggi

RESUME

Riprende l'esecuzione di un programma dopo l'esecuzione di una routine di gestione errori.

TIPO Istruzione

SINTASSI

RESUME *[riga* | NEXT*]*

dove:

riga	etichetta o numero di riga da cui riprendere l'esecuzione;
NEXT	riprende dall'istruzione successiva a quella che ha causato l'errore.

NOTA

Se si omette l'argomento l'esecuzione riprende dall'istruzione che ha causato l'errore.

ESEMPI

RESUME NEXT

RESUME Correggi

RETURN

Ritorna il controllo da una subroutine.

TIPO Istruzione

SINTASSI

RETURN *[riga]*

dove:

riga	etichetta o numero di riga a cui ritornare il controllo; se omesso ritorna all'istruzione successiva al GOSUB.

ESEMPIO

RETURN

RIGHT$

Fornisce il numero indicato di caratteri di una stringa a partire da destra.

TIPO Funzione stringa

SINTASSI

RIGHT$*(stringa,l)*

dove:

stringa	qualsiasi espressione a stringa;
l	numero di caratteri da fornire.

ESEMPIO

```
PRINT RIGHT$("DOTTORE",3)  'scrive ORE
```

RMDIR

Elimina una directory.

TIPO Istruzione DOS

SINTASSI

RMDIR *nome-dir*

dove:

nome-dir espressione a stringa che identifica la directory da eliminare.

ESEMPIO

```
RMDIR "PROVA"
```

RND

Fornisce a caso un numero in semplice precisione compreso tra 0 e 1.

TIPO Funzione matematica

SINTASSI

RND*[(n)]*

dove:

n numero che indica come debba essere generato il numero casuale successivo:

se n<0 lo stesso numero;

se n>0 il numero casuale successivo (puo' essere omesso);

se n=0 l'ultimo numero generato.

ESEMPIO

```
D1 = INT(RND*6) + 1    'Simula il tiro di un dado
```

RSET

Trasferisce i dati dalla memoria al buffer di un file RANDOM (prima dell'istruzione PUT) o allinea a destra il valore di una variabile stringa.

TIPO Istruzione

SINTASSI

RSET *var-stringa* = *espressione-stringa*

dove:

var-stringa variabile stringa o campo di un file RANDOM definito dall'istruzione FIELD;

espressione-stringa qualunque espressione stringa.

ESEMPIO

```
RSET A$ = B$
```

RTRIM$

Elimina gli spazi finali da una stringa.

TIPO Funzione stringa

SINTASSI

RTRIM$*(espressione-stringa)*

dove:

espressione-stringa qualunque espressione stringa.

ESEMPIO

```
A$ = "ROSSI GIOVANNI              "
PRINT RTRIM$(A$)
```

RUN

Esegue il programma corrente o specificato.

TIPO Istruzione

SINTASSI

RUN *[riga | file]*

dove:

riga	etichetta o numero di riga del programma corrente da cui iniziale l'esecuzione; se omesso esegue dalla prima istruzione;
file	nome di un file sorgente BASIC.

ESEMPIO

```
RUN "c:\dos\\test.bas"
```

SCREEN

Fornisce il valore ASCII o l'attributo colore del carattere nella posizione specificata.

TIPO Funzione

SINTASSI

SCREEN(*riga, colonna[, colore]*)

dove:

riga, colonna	coordinate del carattere;
colore	se omesso o **0** fornisce il codice ASCII, se **1** l'attributo di colore.

ESEMPIO

```
PRINT "IL VALORE ASCII E':";SCREEN(10,10)
```

SCREEN

Imposta le caratteristiche dello schermo.

TIPO Istruzione grafica

SINTASSI

SCREEN *mod[, [sw-colore][, [pag-att]*
[, pag-visual]]]

dove:

mod imposta la modalità schermo (si veda la tabella dei valori sulla guida in linea);

sw-colore valore (0 o 1) che in modalità schermo 0 e 1 determina il passaggio tra la visualizzazione a colori o monocromatica;

pag-att numero della pagina di schermo attiva in cui scrivere un testo o un grafico;

pag-visual numero della pagina di schermo attualmente visualizzata.

ESEMPIO

```
SCREEN 1
```

SEEK

Fornisce la posizione corrente nel file, in numero di record per i file RANDOM, in numero di byte per i file sequenziali.

TIPO Funzione

SINTASSI

SEEK(*n-file*)

dove:

n-file numero di un file aperto.

ESEMPIO

```
PRINT "Record successivo:";SEEK(1)
```

SEEK

Imposta il punto del file per la successiva lettura o scrittura.

TIPO Istruzione

SINTASSI

SEEK *n-file, pos*

dove:

n-file	numero di un file aperto;
pos	punto nel file (numero record per i file RANDOM, numero byte per i file sequenziali).

ESEMPIO

```
SEEK #1, 10
```

SELECT CASE

Esegue uno dei blocchi di istruzioni elencati in base al valore di una espressione.

TIPO Istruzione

SINTASSI

SELECT CASE *espr*
CASE *espr-1*
[istruzioni-1]
CASE *espr-2*
[istruzioni-2]
.
.
.
*[*CASE ELSE
[istruzioni-n]]
END SELECT

dove:

espr qualsiasi espressione numerica o a stringa;

espr-1, espr-2:...espr-n espressioni che attivano il relativo blocco di istruzioni quando corrispondono a *espr*;

istruzioni-1,... blocchi di istruzioni.

ESEMPIO

```
SELECT CASE SCELTA%
CASE 15>5
PRINT "Scartato"
CASE 1
PRINT "Prima scelta"
CASE 2 TO 5
PRINT "Seconda scelta"
END SELECT
```

SGN

Indica il segno di un'espressione numerica (1 se positivo, 0 se nullo, -1 se negativo).

TIPO Funzione matematica

SINTASSI

SGN*(espressione-numerica)*

ESEMPIO

```
segno% = SGN(num)
```

SHARED

Consente ad una procedura SUB o FUNCTION l'accesso alle variabili del modulo.

TIPO Istruzione

SINTASSI

SHARED *variabile [*AS *tipo][,...]*

dove:

variabile	nome della variabile del modulo da condividere;
tipo	tipo di dati della variabile (INTEGER, LONG, SINGLE, DOUBLE, STRING).

ESEMPIO

```
SHARED Tot AS SINGLE
```

SHELL

Sospende l'esecuzione di un programma passando il controllo al DOS. Si rientra con il comando EXIT.

TIPO Istruzione

SINTASSI

SHELL *[comando]*

dove:

comando stringa con il comando Dos; se omesso viene visualizzato il PROMPT del DOS.

ESEMPIO

```
SHELL "DIR"
```

SIN

Fornisce il seno di un angolo espresso in radianti.

TIPO Funzione matematica

SINTASSI

SIN*(espressione-numerica)*

dove:

espressione-numerica è un valore o una espressione di qualsiasi tipo.

Per convertire i gradi in radianti, si devono moltiplicare i gradi per (PGRECO/180).

ESEMPIO

```
X = SIN(60 * (PGRECO / 180))
```

SLEEP

Sospende l'esecuzione del programma.

TIPO Istruzione

SINTASSI

SLEEP *[secondi]*

dove:

secondi tempo di sospensione in secondi; se omesso per ripartire occorre premere un tasto.

ESEMPIO

```
SLEEP 15        'si ferma 15 secondi
```

SOUND

Genera un suono all'altoparlante.

TIPO Istruzione

SINTASSI

SOUND *frequenza, durata*

dove:

frequenza	valore numerico (tra 37 e 32.767) della frequenza in Hertz;
durata	valore numerico (tra 0 e 65.535) che indica la durata del suono in numero di scatti dell'orologio del sistema (18,2 scatti al secondo).

ESEMPIO

```
FOR I% = 500 TO 1.000 STEP5
SOUND I%,50
NEXT i%
```

SPACE$

Fornisce una stringa di ***n*** spazi

TIPO Funzione stringa

SINTASSI

SPACE$(*n*)

dove:

n espressione numerica intera.

ESEMPIO

```
FOR I% = 1 TO 15
PRINT SPACE$(I%*2); I%
NEXT I%
```

SPC

Salta ***n*** spazi in un'istruzione PRINT o LPRINT.

TIPO Funzione

SINTASSI

SPC(*n*)

dove:

n espressione numerica intera.

ESEMPIO

```
PRINT "Nome";SPC(8);"Indirizzo";SPC(8);"Telefono
```

SQR

Calcola la radice quadrata.

TIPO Funzione matematica

SINTASSI

SQR(*n*)

dove:

n espressione numerica.

ESEMPIO

```
PRINT SQR(A#)
```

STATIC

Rende una variabile locale all'interno di una funzione DEF FN o di una procedura FUNCTION o SUB e ne conserva il valore tra una chiamata e l'altra.

TIPO Istruzione

SINTASSI

STATIC *variabile [AS tipo][,...]*

dove:

variabile	nome della variabile del modulo da condividere;
tipo	tipo di dati della variabile (INTEGER, LONG, SINGLE, DOUBLE, STRING).

ESEMPIO

```
STATIC totale AS SINGLE
```

STICK

Fornisce le coordinate del Joystick.

TIPO Funzione

SINTASSI

STICK(*n*)

dove:

n	valore numerico da 0 a 3 che indica quale coordinata fornire: 0 x del joystick A 1 y del joystick A 2 x del joystick B 3 y del joystick B

ESEMPIO

```
PRINT STICK(0), STICK(1)
```

STOP

Interrompe l'esecuzione di un programma (F5 la riprende).

TIPO Istruzione

SINTASSI

STOP

ESEMPIO

```
IF A% = 10 THEN STOP
```

STR$

Fornisce un numero in formato stringa.

TIPO Funzione

SINTASSI

STR$*(n)*

dove:

n qualsiasi espressioe numerica.

ESEMPIO

```
A$ = STR$(A)
```

STRIG

Fornisce lo stato del pulsante d'azione del joystick.

TIPO Funzione I/O

SINTASSI

STRIG(*n*)

dove:

n	valore che indica lo stato del joystick da controllare (vedi tabella dei valori sulla guida in linea); se lo stato è vero fornisce -1, se è falso 0.

ESEMPIO

```
IF STRIG(0) THEN GOSUB Via
```

STRIG ON|OFF|STOP

Attiva, disattiva o sospende la gestione degli eventi del joystick.

TIPO Istruzione I/O

SINTASSI

STRIG ON|OFF|STOP

dove:

n valore che indica un pulsante del joystick:

0 pulsante inferiore del joystick A;
2 pulsante inferiore del " B;
4 pulsante superiore del " A;
6 pulsante superiore del " B.

ESEMPIO

STRIG ON

STRING$

Fornisce una stringa di lunghezza specificata costituita dalla ripetizione dello stesso carattere.

TIPO Funzione stringa

SINTASSI

STRING$*(l, car | stringa)*

dove:

l	lunghezza della stringa;
car	codice ASCII del carattere da ripetere;
stringa	qualsiasi espressione a stringa di cui verrà ripetuto il primo carattere.

ESEMPI

```
PRINT STRING$(80,"*")      'Riga di asterischi

PRINT STRING$(80,42)       'Come sopra
```

SUB

Definisce una procedura SUB.

TIPO Istruzione

SINTASSI

SUB *nome[(parametri)] [*STATIC*]*

.
.
.
END SUB

dove:

nome	nome di una procedura SUB;
parametri	una o più variabili che indicano i parametri da passare alla procedura, nella forma: *variabile* AS *tipo [,...]* con *tipo* uguale a INTEGER, LONG, SINGLE, DOUBLE o STRING;
STATIC	indica che i valori delle variabili della procedura SUB debbono essere salvati tra una chiamata a l'altra.

ESEMPIO

SUB ricerca(A$) STATIC

SWAP

Scambia i valori di due variabili.

TIPO Istruzione

SINTASSI

SWAP *var1, var2*

dove:

var1,var2 variabili dello stesso tipo.

ESEMPIO

SWAP A%, B%

SYSTEM

Chiude i file aperti e ritorna il controllo al DOS.

TIPO Istruzione

SINTASSI

SYSTEM

TAB

Sposta il cursore di testo nel punto in cui iniziare a scrivere.

TIPO Funzione I/O

SINTASSI

TAB*(col)*

dove:

col	numero della colonna a cui iniziare a scrivere.

ESEMPIO

```
PRINT TAB(30); TITOLO$
```

TAN

Fornisce la tangente di un angolo espresso in radianti.

TIPO Funzione matematica

SINTASSI

TAN*(espressione-numerica)*

dove:

espressione-numerica è un valore o una espressione numerica di qualsiasi tipo. Per convertire i gradi in radianti si moltiplicano i gradi per (PIGRECO/180).

ESEMPIO

```
X = TAN(A%)
```

TIME$

Fornisce una stringa contenenente l'ora corrente.

TIPO Funzione DOS.

SINTASSI

TIME$

Fornisce l'ora nel formato: OO:MM:SS

ESEMPIO

```
PRINT TIME$
```

TIME$

Imposta l'ora corrente.

TIPO Istruzione DOS

SINTASSI

TIME$ = *stringa-ora*

dove:

stringa-ora espressione stringa in uno dei seguenti formati:

OO imposta l'ora (minuti e secondi a 0)
OO:MM imposta lo'ora e i minuti (secondi a 0)
OO:MM:SS imposta l'ora, i minuti e i secondi.

ESEMPIO

```
TIME$ = "10:30"
```

TIMER

Fornisce il numero di secondi trascorsi dalla mezzanotte.

TIPO Funzione

SINTASSI

TIMER

ESEMPIO

```
RANDOMIZE TIMER
```

TIMER ON|OFF|STOP

Attiva, disattiva o sospende la gestione degli eventi del temporizzatore.

TIPO Istruzione

SINTASSI

TIMER ON|OFF|STOP
ON TIMER(n)GOSUB riga

dove:

n	numero di secondi trascorsi prima che il controllo passi a
riga	etichetta o numero di riga della subroutine di gestione degli eventi a cui passare il controllo.

ESEMPIO

```
ON TIMER(5)GOSUB Alfa
TIMER ON
```

TRON, TROFF

Attiva o disattiva la traccia delle istruzioni eseguite in un programma.

TIPO Istruzioni

SINTASSI

TRON

TROFF

TYPE

Definisce un tipo di dati contenente uno o più elementi.

TIPO Istruzione

SINTASSI

TYPE *tipo-dato*
nome-elem AS *nome-tipo*
.
.
.
END TYPE

dove:

tipo-dato nome del tipo di dati;

nome-elem un elemento del tipo di dati;

nome-tipo uno dei seguenti tipi: INTEGER, LONG, SINGLE, DOUBLE o STRING**n* (stringa lunga *n*).

ESEMPIO

```
TYPE Carta
COLORE AS STRING*9
VALORE AS INTEGER
END TYPE
```

UBOUND

Fornisce il limite superiore della dimensione indicata di una matrice.

TIPO Funzione

SINTASSI

UBOUND(*matrice[, dim]*)

dove:

matrice	nome della matrice;
dim	numero intero che indica di quale dimensione della matrice fornire il limite superiore; opzionale per vettori (matrici unidimensionali).

ESEMPIO

```
DIM A(1 TO 100,0 TO 50,-3 TO 4)
UBOUND (A,1)                'fornisce 100
UBOUND (A,2)                'fornisce 50
UBOUND (A,3)                'fornisce 4
```

UCASE$

Converte in maiuscolo tutti i caratteri di una stringa.

TIPO Funzione stringa

SINTASSI

UCASE$*(stringa)*

dove:

stringa qualsiasi espressione a stringa.

ESEMPIO

```
PRINT UCASE$ (nome$)
```

UEVENT

Attiva, disattiva o sospende la gestione di un evento definito dall'utente.

TIPO Istruzione

SINTASSI

UEVENT ON|OFF|STOP

ESEMPIO

UEVENT ON

VAL

Riconverte un numero in formato stringa in numero.

TIPO Funzione

SINTASSI

VAL(*n-stringa*)

dove:

n-stringa un numero in formato stringa.

ESEMPIO

```
a = VAL(a$)
```

VARPTR, VARSEG

Forniscono l'indirizzo di offset e di segmento di una variabile.

TIPO Funzione

SINTASSI

VARPTR*(nomevar)*

VARSEG*(nomevar)*

dove:

nomevar qualunque variabile.

ESEMPIO

```
DEF SEG = VARSEG(cubo(1))
```

VARPTR$

Fornisce una rappresentazione a stringa dell'indirizzo di una variabile, ad uso delle istruzioni DRAW e PLAY.

TIPO Funzione stringa

SINTASSI

VARPTR$(*nomevar*)

dove:

nomevar qualunque variabile.

ESEMPIO

```
PLAY "X" + VARPTR$(A$)
```

VIEW

Definisce la finestra di schermo per la visualizzazione di grafici.

TIPO Istruzione grafica

SINTASSI

VIEW *[[SCREEN] (x1, y2) - (x2, y2)[, [colore] [, bordo]]]*

dove:

SCREEN indica che le coordinate sono relative allo schermo;

(x1, y1) - (x2, y2) coordinate degli angoli diametralmente opposti della finestra;

colore attributo di colore di riempimento;

bordo attributo di colore del bordo.

ESEMPIO

```
VIEW(10, 20) - (200, 150), 1
```

VIEW PRINT

Definisce la finestra di schermo per la visualizzazione di testo.

TIPO Istruzione

SINTASSI

VIEW PRINT *[riga-i* TO *riga-f]*

dove:

riga-i, riga-f indirizzi di inizio e fine della nuova finestra di testo.

ESEMPIO

```
VIEW PRINT 12 TO 24
```

WAIT

Sospende l'esecuzione del programma durante il controllo dello stato di una porta di input.

TIPO Istruzione

SINTASSI

WAIT *n-porta, espressione-and [, espressione-xor]*

dove:

n-porta numero della porta;

espressione-and espressione intera che viene unita ai dati arrivati alla porta con una operazione AND;

espressione-xor espressione intera che viene unita ai dati arrivati alla porta con una operazione XOR.

ESEMPIO

```
WAIT &H20,1
```

WHILE...WEND

Esegue una serie di istruzioni fino a quando la condizione specificata resta vera.

TIPO Istruzione

SINTASSI

WHILE *condizione*
[blocco istruzioni]
WEND

dove:

condizione espressione condizionale;

ESEMPIO

```
WHILE n% > 0
n% = n% - 1
PRINT testo$
WEND
```

WIDTH

Imposta l'ampiezza della riga di output per una periferica (stampante) o modifica il numero di righe e colonne visualizzate sullo schermo.

TIPO Istruzione

SINTASSI

WIDTH *[colonne][, righe]*

WIDTH *nfile* | *periferica, dim-col*

WIDTH LPRINT *dim-col*

dove:

colonne, righe numero di colonne e righe visualizzate sullo schermo (a seconda dell'adattatore video);

nfile | *periferica* numero del file o nome della periferica (LPT1);

dim-col ampiezza della riga di output.

NOTA

WIDTH LPRINT imposta l'ampiezza di riga della stampante a file aperto, mentre WIDTH *nfile|periferica* viene attivato dalla OPEN.

ESEMPIO

```
WIDTH #1, 3
```

WINDOW

Definisce le dimensioni logiche della finestra corrente.

TIPO Istruzione

SINTASSI

WINDOW *[[*SCREEN*] (x1,y1) - (x2,y2)]*

dove:

SCREEN inverte la normale direzione delle coordinate y, in modo che i valori di y aumentino dall'altro verso il basso;

(x1,y1) - (x2,y2) coordinate degli angoli diametralmente opposti della finestra.

ESEMPIO

```
WINDOW (-160/I%, -100/I%) - (160/I%, 100/I%)
```

WRITE

Scrive i dati sullo schermo o su un file sequenziale.

TIPO Istruzione I/O

SINTASSI

WRITE *[n-file,] elenco-espressioni*

dove:

n-file numero di un file sequenziale aperto; se omesso scrive i dati sullo schermo;

elenco-espressioni una o più variabili o espressioni, separate da virgola, i cui valori vengono scritti

ESEMPIO

```
WRITE #1, Nome$, Indirizzo$, Telefono$
```

RINGRAZIAMENTI

Questo manuale e' stato redatto e stampato con l'ausilio di strumenti informatici, con particolare riferimento al software applicativo MS-WINDOWS, applicazioni WORD e PAINTBRUSH, quest'ultimo utilizzato per realizzare la maggior parte delle figure che illustrano il testo.

La Microsoft, al pari di altre societa' che sviluppano software, si prodiga da anni nella ricerca, allo scopo di migliorare le prestazioni dei prodotti e al tempo stesso di rendere piu' amichevole e facilmente utilizzabile lo strumento piu' affascinante che la tecnologia abbia mai creato: il PERSONAL COMPUTER.